Representation Learning on Complex Data

Dissertation zur Erlangung des Doktorgrades
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

vorgelegt von
Julian Busch
aus Hamburg

München, den 16.03.2021

Tag der Einreichung:	16.03.2021
Erstgutachter:	Prof. Dr. Thomas Seidl Ludwig-Maximilians-Universität München
Zweitgutachter:	Prof. Dr. Emmanuel Müller Technische Universität Dortmund
Drittgutachter:	Prof. Dr. Kristian Kersting Technische Universität Darmstadt
Vorsitz:	Prof. Dr. Albrecht Schmidt Ludwig-Maximilians-Universität München
Tag der Disputation:	29.07.2021

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.2011, § 8, Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

München, den 07.09.2021

Julian Busch

Contents

Abstract

Machine learning has enabled remarkable progress in various fields of research and application in recent years. The primary objective of machine learning consists of developing algorithms that can learn and improve through observation and experience. Machine learning algorithms learn from data, which may exhibit various forms of complexity, which pose fundamental challenges. In this thesis, we address two major types of data complexity: First, data is often inherently connected and can be modeled by a single or multiple graphs. Machine learning methods could potentially exploit these connections, for instance, to find groups of similar users in a social network for targeted marketing or to predict functional properties of proteins for drug design. Secondly, data is often high-dimensional, for instance, due to a large number of recorded features or induced by a quadratic pixel grid on images. Classical machine learning methods perennially fail when exposed to high-dimensional data as several key assumptions cease to be satisfied.

Therefore, a major challenge associated with machine learning on graphs and high-dimensional data is to derive meaningful representations of this data, which allow models to learn effectively. In contrast to conventional manual feature engineering methods, representation learning aims at automatically learning data representations that are particularly suitable for a specific task at hand. Driven by a rapidly increasing availability of data, these methods have celebrated tremendous success for tasks such as object detection in images and speech recognition. However, there is still a considerable amount of research work to be done to fully leverage such techniques for learning on graphs and high-dimensional data.

In this thesis, we address the problem of learning meaningful representations for highly-effective machine learning on complex data, in particular, graph data and high-dimensional data. Additionally, most of our proposed methods are highly scalable, allowing them to learn from massive amounts of data. While we address a wide range of general learning problems with different modes of supervision, ranging from unsupervised problems on unlabeled data to (semi-)-supervised learning on annotated data sets, we evaluate our models on specific tasks from fields such as social network analysis, information security, and computer vision.

The first part of this thesis addresses representation learning on graphs. While existing graph neural network models commonly perform synchronous message passing between nodes and thus struggle with long-range dependencies and efficiency issues, our first proposed method performs fast asynchronous message passing and, therefore, supports adaptive and efficient learning and additionally scales to large graphs. Another contribution consists of a novel graph-based approach to malware detection and classification based on network traffic. While existing methods classify individual network flows between two endpoints, our algorithm collects all traffic in a monitored network within a specific time frame and builds a communication graph, which is then classified using a novel graph neural network model. The developed model can be generally applied to further graph classification or anomaly detection tasks. Two further contributions challenge a common assumption made by graph learning methods, termed homophily, which states that nodes with similar properties are usually closely connected in the graph. To this end, we develop a method that predicts node-level properties leveraging the distribution of class labels appearing in the neighborhood of the respective node. That allows our model to learn general relations between a node and its neighbors, which are not limited to homophily. Another proposed method specifically models structural similarity between nodes to model different roles, for instance, influencers and followers in a social network. In particular, we develop an unsupervised algorithm for deriving node descriptors based on how nodes spread probability mass to their neighbors and aggregate these descriptors to represent entire graphs.

The second part of this thesis addresses representation learning on high-dimensional data. Specifically, we consider the problem of clustering high-dimensional data, such as images, texts, or gene expression profiles. Classical clustering algorithms struggle with this type of data since it can usually not be assumed that data objects will be similar w.r.t. all attributes, but only within a particular subspace of the full-dimensional ambient space. Subspace clustering is an approach to clustering high-dimensional data based on this assumption. While there already exist powerful neural network-based subspace clustering methods, these methods commonly suffer from scalability issues and lack a theoretical foundation. To this end, we propose a novel metric learning approach to subspace clustering, which can provably recover linear subspaces under suitable assumptions and, at the same time, tremendously reduces the required number of model parameters and memory compared to existing algorithms.

Zusammenfassung

Maschinelles Lernen hat in den letzten Jahren bemerkenswerte Fortschritte in verschiedenen Forschungs- und Anwendungsbereichen ermöglicht. Das primäre Ziel des maschinellen Lernens besteht darin, Algorithmen zu entwickeln, die durch Beobachtung und Erfahrung lernen und sich verbessern können. Algorithmen des maschinellen Lernens lernen aus Daten, die verschiedene Formen von Komplexität aufweisen können, was grundlegende Herausforderungen mit sich bringt. Im Rahmen dieser Dissertation werden zwei Haupttypen von Datenkomplexität behandelt: Erstens weisen Daten oft inhärente Verbindungen, die durch einen einzelnen oder mehrere Graphen modelliert werden können. Methoden des maschinellen Lernens können diese Verbindungen potenziell ausnutzen, um beispielsweise Gruppen ähnlicher Nutzer in einem sozialen Netzwerk für gezieltes Marketing zu finden oder um funktionale Eigenschaften von Proteinen für das Design von Medikamenten vorherzusagen. Zweitens sind die Daten oft hochdimensional, z. B. aufgrund einer großen Anzahl von erfassten Merkmalen oder bedingt durch ein quadratisches Pixelraster auf Bildern. Klassische Methoden des maschinellen Lernens versagen immer wieder, wenn sie hochdimensionalen Daten ausgesetzt werden, da mehrere Schlüsselannahmen nicht mehr erfüllt sind.

Daher besteht eine große Herausforderung beim maschinellen Lernen auf Graphen und hochdimensionalen Daten darin, sinnvolle Repräsentationen dieser Daten abzuleiten, die es den Modellen ermöglichen, effektiv zu lernen. Im Gegensatz zu konventionellen manuellen Feature-Engineering-Methoden zielt Representation Learning darauf ab, automatisch Datenrepräsentationen zu lernen, die für eine bestimmte Aufgabenstellung besonders geeignet sind. Angetrieben durch eine rasant steigende Datenverfügbarkeit haben diese Methoden bei Aufgaben wie der Objekterkennung in Bildern und der Spracherkennung enorme Erfolge gefeiert. Es besteht jedoch noch ein erheblicher Forschungsbedarf, um solche Verfahren für das Lernen auf Graphen und hochdimensionalen Daten voll auszuschöpfen.

Diese Dissertation beschäftigt sich mit dem Problem des Lernens sinnvoller Repräsentationen für hocheffektives maschinelles Lernen auf komplexen Daten, insbesondere auf Graphen und hochdimensionalen Daten. Zusätzlich sind die meisten hier vorgeschlagenen Methoden hoch skalierbar, so dass sie aus großen Datenmengen lernen können.

Obgleich eine breite Palette von allgemeinen Lernproblemen mit verschiedenen Arten der Überwachung adressiert wird, die von unüberwachten Problemen auf unannotierten Daten bis hin zum (semi-)überwachten Lernen auf annotierten Datensätzen reichen, werden die vorgestellten Metoden anhand spezifischen Anwendungen aus Bereichen wie der Analyse sozialer Netzwerke, der Informationssicherheit und der Computer Vision evaluiert.

Der erste Teil der Dissertation befasst sich mit dem Representation Learning auf Graphen. Während existierende neuronale Netze für Graphen üblicherweise eine synchrone Nachrichtenübermittlung zwischen den Knoten durchführen und somit mit langreichweitigen Abhängigkeiten und Effizienzproblemen zu kämpfen haben, führt die erste hier vorgeschlagene Methode eine schnelle asynchrone Nachrichtenübermittlung durch und unterstützt somit adaptives und effizientes Lernen und skaliert zudem auf große Graphen. Ein weiterer Beitrag besteht in einem neuartigen graphenbasierten Ansatz zur Malware-Erkennung und -Klassifizierung auf Basis des Netzwerkverkehrs. Während bestehende Methoden einzelne Netzwerkflüsse zwischen zwei Endpunkten klassifizieren, sammelt der vorgeschlagene Algorithmus den gesamten Verkehr in einem überwachten Netzwerk innerhalb eines bestimmten Zeitraums und baut einen Kommunikationsgraphen auf, der dann mithilfe eines neuartigen neuronalen Netzes für Graphen klassifiziert wird. Das entwickelte Modell kann allgemein für weitere Graphenklassifizierungs- oder Anomalieerkennungsaufgaben eingesetzt werden. Zwei weitere Beiträge stellen eine gängige Annahme von Graphen-Lernmethoden in Frage, die so genannte Homophilie-Annahme, die besagt, dass Knoten mit ähnlichen Eigenschaften in der Regel eng im Graphen verbunden sind. Zu diesem Zweck wird eine Methode entwickelt, die Eigenschaften auf Knotenebene vorhersagt, indem sie die Verteilung der annotierten Klassen in der Nachbarschaft des jeweiligen Knotens nutzt. Das erlaubt dem vorgeschlagenen Modell, allgemeine Beziehungen zwischen einem Knoten und seinen Nachbarn zu lernen, die nicht auf Homophilie beschränkt sind. Eine weitere vorgeschlagene Methode modelliert strukturelle Ähnlichkeit zwischen Knoten, um unterschiedliche Rollen zu modellieren, zum Beispiel Influencer und Follower in einem sozialen Netzwerk. Insbesondere entwickeln wir einen unüberwachten Algorithmus zur Ableitung von Knoten-Deskriptoren, die darauf basieren, wie Knoten Wahrscheinlichkeitsmasse auf ihre Nachbarn verteilen, und aggregieren diese Deskriptoren, um ganze Graphen darzustellen.

Der zweite Teil dieser Dissertation befasst sich mit dem Representation Learning auf hochdimensionalen Daten. Konkret wird das Problem des Clusterns hochdimensionaler Daten, wie z. B. Bilder, Texte oder Genexpressionsprofile, betrachtet. Klassische Clustering-Algorithmen haben mit dieser Art von Daten zu kämpfen, da in der Regel nicht davon ausgegangen werden kann, dass die Datenobjekte in Bezug auf alle Attribute ähnlich sind, sondern nur innerhalb eines bestimmten Unterraums des volldimensionalen Datenraums. Das Unterraum-Clustering ist ein Ansatz zum Clustern hochdimensionaler Daten, der auf dieser Annahme basiert. Obwohl es bereits leistungsfähige, auf neuronalen Netzen basierende Unterraum-Clustering-Methoden gibt, leiden diese Methoden im Allgemeinen unter Skalierbarkeitsproblemen und es fehlt ihnen an einer theoretischen Grundlage. Zu diesem Zweck wird ein neuartiger Metric Learning Ansatz für das Unterraum-Clustering vorgeschlagen, der unter geeigneten Annahmen nachweislich lineare Unterräume detektieren kann und gleichzeitig die erforderliche Anzahl von Modellparametern und Speicher im Vergleich zu bestehenden Algorithmen enorm reduziert.

Acknowledgments

My dissertation would not have been possible without the guidance and support, both scientific and personal, I have received from numerous people over the recent years. I am deeply grateful and wish to profoundly thank everyone who accompanied me on my way.

First and foremost, I wish to thank my doctoral advisor Prof. Dr. Thomas Seidl. You gave me the opportunity to take on this challenge, funded my position, and procured an exceptionally open, friendly, and productive working atmosphere as the head of our group. This work would not have been possible without the countless fruitful discussions and your continued guidance, which allowed me to grow as a scientist.

I also wish to thank Prof. Dr. Emmanuel Müller and Prof. Dr. Kristian Kersting for their interest in my work and their willingness to examine my dissertation as secondary referees.

Deepest gratitude also goes to all of my former and current colleagues at the Database Systems and Data Mining group and LMU Munich. Thank you for all your support, all the fun, countless lunch and coffee breaks, and for sharing all the ups and downs throughout the journey. I especially want to thank Felix Borutta, Evgeniy Faerman, Daniyal Kazempour, Anna Beer, Max Berrendorf, Sebastian Schmoll, Florian Richter, Maximilian Hünemörder, Ludwig Zellner, Prof. Dr. Matthias Schubert, and Prof. Dr. Peer Kröger. It was a pleasure working with you all.

I further want to thank Philipp Schaefer and Prof. Dr. Volker Tresp from Siemens, who gave me the opportunity for my internship. Special thanks also go to Dr. Jiaxing Pi and Dr. Anton Kocheturov for supervising my work and to all other colleagues and interns at Siemens, who made my time there an incredibly productive and pleasurable experience.

A special thanks also go to Susanne Grienberger for supporting me in all administrative matters, and to Franz Krojer for all technical support, and for keeping our computing infrastructure from breaking down.

Finally, I wish to express my deepest gratitude to my family and friends, especially my parents, Roland and Bettina, my brothers, Daniel and Florian, and my dearest partner, Jing. Your continued support and encouragement kept me moving forward all this time.

Chapter 1

Introduction

Our knowledge springs from two fundamental sources of the mind; the first is the capacity of receiving representations (receptivity for impressions), the second is the power of knowing an object through these representations (spontaneity of concepts).

Immanuel Kant

1.1 Thesis Statement

Machine learning has enabled remarkable progress in various fields, including science, production and automation in industry, and data analytics in corporations, finance, mobility, and healthcare. It has thus exerted a significant impact on the economy and society as a whole and became a driving factor for its development. Machine learning aims at developing algorithms that are able to learn and improve through observation and experience. As such, machine learning methods are fueled by data, which comes in a multitude of different forms, ranging from scientific data sets to images, written and spoken natural language, and data accumulated in social networks. An exponential increase in data availability in recent years has made this transformation possible, but at the same time, machine learning methods are faced with various forms of data complexity, which pose fundamental challenges.

This thesis addresses two major types of data complexity: First, data is often inherently connected and can be modeled by a single or multiple graphs. Machine learning methods could potentially exploit these connections, for instance, to find groups of similar users in a social network for targeted marketing or to predict functional properties of

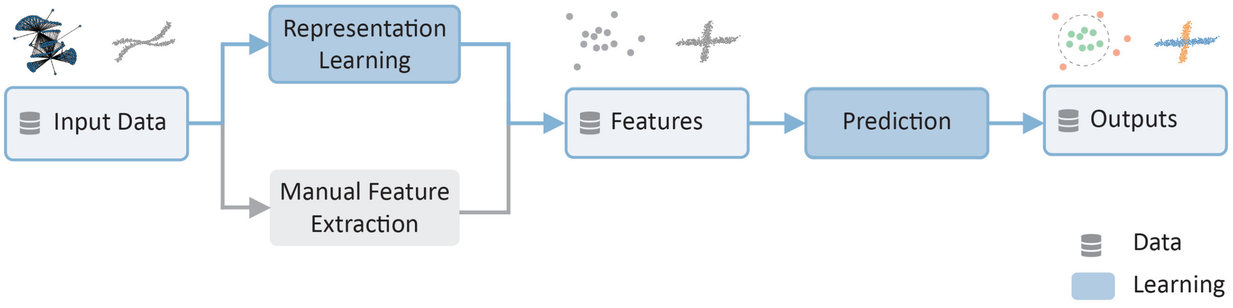


Figure 1.1: A machine learning algorithm requires expressive features to provide accurate predictions from the input data. Conventional approaches involve manual feature extraction or engineering, which is challenging even for domain experts and is not guaranteed to lead to highly accurate predictions. Representation learning algorithms, on the other hand, learn expressive features together with the prediction algorithm, such that the learned features are optimized to be maximally useful to the algorithm. In this thesis, we advance representation learning on graphs and high-dimensional data. The illustrations on the top sketch a graph anomaly detection problem for malware detection that we addressed in [BKTS21] and a non-linear subspace clustering problem that we treated in [BFSS20].

proteins for drug design. Secondly, data is often high-dimensional, for instance, due to a large number of recorded features, or induced by a quadratic pixel grid on images or bag-of-words representations for texts which count occurrences of all words contained in a dictionary. Classical machine learning methods perennially fail when exposed to high-dimensional data as several key assumptions cease to be satisfied.

A major challenge associated with machine learning on graphs and high-dimensional data is, therefore, to derive meaningful representations of this data, which allow models to learn effectively. Conventionally, manual feature extraction or engineering is employed as a pre-processing step, connected upstream to the actual learning task. However, it is usually very challenging even for domain experts to determine representations that are particularly meaningful or useful for a specific task. Driven by a rapidly increasing availability of data, deep learning methods have celebrated tremendous success at automatically learning representations that are optimized end-to-end towards the objective of a particular learning task instead of being pre-computed upstream. While deep learning has remarkably advanced state-of-the-art for tasks such as object detection in images and speech recognition, there is still a considerable amount of research work to be done to leverage representation learning techniques for learning on graphs and high-dimensional data.

The overarching research objective of this thesis consists in the development of algorithms that are able to learn meaningful representations for highly effective machine learning on complex data, in particular graphs and high-dimensional data. A visual summary of this thesis statement is provided in Figure 1.1. Additionally, most of our proposed methods are highly scalable, allowing them to learn from massive amounts of data. While we address a wide range of general learning problems with different modes of supervision, ranging from unsupervised problems on unlabeled data to (semi-)-supervised learning on annotated data sets, we evaluate our models on specific tasks from fields such as social network analysis, information security, and computer vision.

1.2 Summary of Contributions

The first part of this thesis addresses representation learning on graphs. Common learning problems on graphs include unsupervised clustering and anomaly detection and (semi-)supervised classification of graph nodes or entire graphs, for instance, to find groups of similar social network users or to predict user interests or properties of whole communities.

In recent years, graph neural network models have been established as state-of-the-art approaches to these problems. While existing graph neural network models commonly perform synchronous message passing between nodes and thus struggle with long-range dependencies and efficiency issues, our first proposed method performs fast asynchronous message passing and, therefore, supports adaptive and efficient learning and additionally scales to large graphs. Empirically, our method significantly improves state-of-the-art on several semi-supervised document classification tasks.

Another contribution consists of a novel graph-based approach to malware detection and classification based on network traffic. While existing methods classify individual network flows between two endpoints, our algorithm collects all traffic in a monitored network within a specific time frame and builds a communication graph, which is then classified using a novel graph neural network model. Using this additional relational information enables our algorithm to significantly boost detection and classification performance on a benchmark dataset. Further, the developed algorithm can be generally applied to further graph classification or anomaly detection tasks.

Two further contributions challenge a common assumption made by graph learning methods, termed homophily, which states that nodes with similar properties are usually closely connected in the graph. To this end, we develop a method that predicts node-level properties leveraging the distribution of class labels appearing in the neighborhood of the respective node. This allows our model to learn general relations between a node and its neighbors, which are not limited to homophily to improve prediction accuracy on data sets for document classification, interest prediction in a social network, and genre prediction in a movie-actor network.

Another proposed method specifically models structural similarity between nodes to model different roles, for instance, influencers and followers in a social network. In particular, we develop an unsupervised algorithm for deriving node descriptors based on how nodes spread probability mass to their neighbors and aggregate these descriptors to represent entire graphs. In addition to improving downstream graph classification accuracy on airline networks, biological and social network datasets, our method is additionally highly efficient and scalable.

The second part of this thesis addresses representation learning on high-dimensional data. Specifically, we consider the problem of clustering high-dimensional data, such as images, texts, or gene expression profiles. Classical clustering algorithms struggle with this type of data since it can usually not be assumed that data objects will be similar w.r.t. all attributes, but only within a particular subspace of the full-dimensional ambient space. Subspace clustering is an approach to clustering high-dimensional data based on this assumption.

While there already exist powerful neural network-based subspace clustering methods, these methods commonly suffer from scalability issues and lack a theoretical foundation. To this end, we propose a novel metric learning approach to subspace clustering, which can provably recover linear subspaces under suitable assumptions and, at the same time, tremendously reduces the required number of model parameters and memory compared to existing algorithms. In combination with an autoencoder, our method is able to overcome the linearity assumption by automatically learning representations that are particularly well-suited for linear subspace clustering. We evaluate our model on an image clustering task where we achieve competitive clustering accuracy at a memory reduction of multiple orders of magnitude, allowing our model to scale to data sets of sizes that have been out of reach of existing algorithms.

1.3 Thesis Outline

The remainder of the thesis is structured as follows. Chapter 2 first provides necessary background from the field of machine learning by introducing some of its fundamental concepts. Based on this foundation, Chapter 3 briefly introduces the research area of machine learning on graphs, provides an overview of the current state of the art, and presents the contributions of this thesis to this field. Chapter 4 addresses machine learning on high-dimensional data. Again, we will first provide an overview of the field and current state of the art and subsequently present the contributions to this field provided by this thesis. Chapter 5 concludes with a summary and an outlook to potential future work. Previous publications contributing to this cumulative dissertation are listed in Appendix A, along with a clarification of the individual authors' contributions.

Chapter 2

Fundamentals of Machine Learning

Machine learning is concerned with the study algorithms that are able to learn and improve from observation and experience and aims to design and analyze such algorithms. As a field of research, it is closely related to other areas, including statistics and mathematical optimization, which provide some fundamental concepts and methods. Data mining is another closely related field concerned with the extraction of either regular or irregular patterns from potentially large volumes of data but usually doesn't involve learning or adaptation. Machine learning, on the other hand, instead focuses on solving specific tasks by learning from data. It is further related to artificial intelligence, which focuses on intelligent agents that interact with some environment, whereas machine learning also considers passive settings not involving any active agents. Applications for machine learning methods are manifold and have contributed to driving developments in science, industrial production and automation, and data analytics in corporations, finance, healthcare, and mobility.

A distinctive feature of machine learning algorithms is that they perform tasks without being explicitly told which exact steps to take. For instance, consider the problem of detecting a cat in an RGB-image. One could think of a set of different rules to decide whether the image shows a cat, for instance, by looking for certain key features such as ears or whiskers. However, there is a potentially infinite variety of cats' potential appearances, and parameters such as position, rotation, illumination, and occlusion in the image are subject to change. Thus, it is virtually impossible to solve this detection task by deriving a set of specific rules. Instead, machine learning algorithms automatically extract general rules on how to perform this detection task by learning from data, in this case, example images showing cats.

2.1 Machine Learning Tasks

Different machine learning tasks can be categorized based on the amount of supervision available to the algorithm. In a *supervised learning* setting, the available data consists of pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$ with input space $\mathcal{X}$ and output space $\mathcal{Y}$, i.e., each data point x is annotated with a label y representing the desired output or prediction for that data point. Machine learning algorithms often assume real vectors as input, i.e., $\mathcal{X} = \mathbb{R}^d$, but inputs could also correspond to more complex objects, such as images or graphs. The algorithm's task is to learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ from a given set of training pairs, which produces correct predictions. A typical supervised learning task is *classification*, where $\mathcal{Y} = \{0, \ldots, K\}$ is a set of discrete class labels and the algorithm needs to predict the correct class for each input. Classification problems with $K = 2$ are referred to as *binary* classification. If $K > 2$, the problem called a *multi-class* classification problem. In case there are multiple possible labels to predict for an input, the problem is referred to as *multi-label* classification. Examples for supervised learning tasks include image recognition, where images are classified to represent different objects, automatic translation of natural language, where sentences of source and target language are paired, and classification of protein molecule graphs according to their metabolic function.

In contrast to supervised learning, in an *unsupervised learning* setting, only the inputs x are given, and there are no target annotations available to the algorithm. The algorithm's task is to detect novel concepts which are not yet captured by any available annotation. A common unsupervised task is *clustering* where the algorithm's job is to partition the inputs into meaningful groups, such that data objects in the same group are similar to each other and objects in different groups are dissimilar to each other. These groups may represent different concepts present in the data, for instance, in image clustering, different objects that are represented by an image. The inverse task of finding objects that are different from the typical behavior observed in the data, rather than regular concepts in the data, is referred to as *anomaly detection*. In a social network, users could be clustered to detect communities of similar users for targeted marketing. On the other hand, anomaly detection could reveal malicious fake users or bots, which behave differently from regular users.

If annotations are only available for a relatively small portion of the data, the setting is referred to as *semi-supervised learning*. In this case, the learning task is a supervised one, but in addition to a relatively small number of labeled pairs, a larger amount of unlabeled data is available. This unlabeled data can potentially improve learning performance since it can reveal more insight into the underlying structure of the data. For instance, the task of semi-supervised node classification in graphs will be introduced in Chapter 3 and allows for, e.g., predicting users interests in social networks for targeted marketing or classifying proteins according to their metabolic functions based on their interactions as modeled by an interaction graph.

The contributions presented in this thesis include algorithms that can be employed to solve machine learning tasks from all of the above categories.

2.2 Inductive Bias

For a machine learning algorithm to be successful, it is essential that the algorithm does not merely memorize the given examples but is actually able to generalize meaningful concepts from the data. To be able to do so, it is necessary to make some assumptions on how to extract such concepts. Such assumptions are also referred to as *inductive bias*, and different machine learning algorithms or classes of algorithms differ by their inductive bias. For instance, different neural network architectures, such as dense, convolutional, or recurrent networks, model different types of spatial or sequential relationships in the data. In graph machine learning, the homophily assumption could be employed as an inductive bias to reflect that similar predictions should be made for users closely connected in a social network. In high-dimensional clustering, a typical assumption is that data points in one cluster lie within the same subspace of the data space.

While it is always possible to design an arbitrarily complex machine learning algorithm that solves a particular task without error on a given dataset, such an algorithm would most probably not perform well on new unseen data. This behavior is also referred to as *over-fitting*. On the other hand, if the algorithm is not complex enough, it is not able to learn useful concepts and is thus prone to *under-fitting*. Thus, machine learning algorithms need to be carefully designed to ensure that relevant patterns present in the data can be captured and, at the same time, that the algorithm is neither too complex nor too simplistic. This can be achieved by selecting an appropriate inductive bias.

Advantages of the algorithms contributed in this thesis over existing algorithms are partly due to their respective inductive bias, which attempts to capture the nature of the considered problem as accurately as possible.

2.3 Neural Networks

For many machine learning algorithms, learning is equivalent to minimizing a particular cost function w.r.t. unknown model parameters. In this section, we illustrate this with *neural network* models, which form the basis of many of the algorithms contributed in this thesis. Neural networks form a class of very versatile models that can be employed to solve a wide variety of tasks on different types of data by designing different network architectures that capture specific inductive biases. Further, neural network training and inference processes are commonly amenable to massive parallelization since they usually

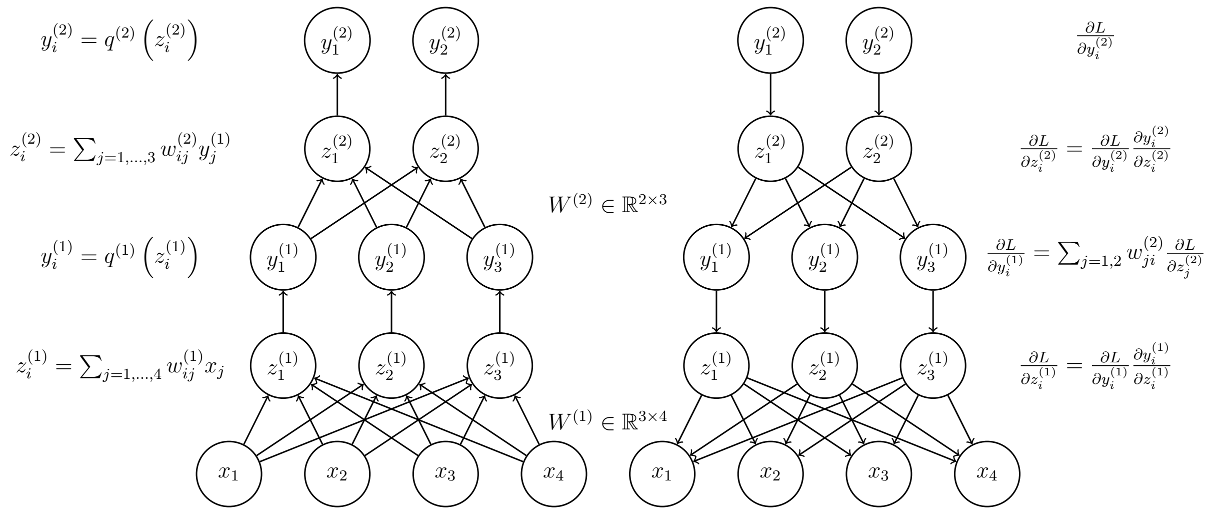

$y_i^{(2)} = q^{(2)}\left(z_i^{(2)}\right)$
$z_i^{(2)} = \sum_{j=1,\ldots,3} w_{ij}^{(2)} y_j^{(1)}$
$y_i^{(1)} = q^{(1)}\left(z_i^{(1)}\right)$
$z_i^{(1)} = \sum_{j=1,\ldots,4} w_{ij}^{(1)} x_j$
$W^{(2)} \in \mathbb{R}^{2\times 3}$
$W^{(1)} \in \mathbb{R}^{3\times 4}$
$\frac{\partial L}{\partial y_i^{(2)}}$
$\frac{\partial L}{\partial z_i^{(2)}} = \frac{\partial L}{\partial y_i^{(2)}}\frac{\partial y_i^{(2)}}{\partial z_i^{(2)}}$
$\frac{\partial L}{\partial y_i^{(1)}} = \sum_{j=1,2} w_{ji}^{(2)} \frac{\partial L}{\partial z_j^{(2)}}$
$\frac{\partial L}{\partial z_i^{(1)}} = \frac{\partial L}{\partial y_i^{(1)}}\frac{\partial y_i^{(1)}}{\partial z_i^{(1)}}$

Figure 2.1: Example of a two-layer MLP. For simplicity, the bias vectors have been omitted. During the forward pass (left-hand side), a 4-dim. input vector x is first mapped to a 3-dim. latent vector $y^{(1)}$ and subsequently to a 2-dim. output vector $y^{(2)}$. During the backward pass (right-hand side), the corresponding gradients w.r.t. the loss function are propagated in the backward direction to update the parameters matrices $W^{(1)}$ and $W^{(2)}$.

correspond to mostly matrix and vector algebra operations that can be mapped well to GPU architectures and are thus able to leverage vast computational resources.

A basic neural network often used as a building block for more complex neural network architectures is the *Multi-Layer Perceptron (MLP)*. An MLP consists of multiple subsequent layers, each of which can be formally written as

$$z^{(k)} = W^{(k)} y^{(k-1)} + b^{(k)} \tag{2.1}$$

$$y^{(k)} = q^{(k)}\left(z^{(k)}\right), \tag{2.2}$$

where the output $y^{(k-1)} \in \mathbb{R}^{h_{k-1}}$ of the $(k-1)$-th layer corresponds to the input of the k-th layer and the input to the first layer is the original input vector $x \in \mathbb{R}^d$. The $k-$th layer applies an affine transformation with *weight matrix* $W \in \mathbb{R}^{h_k \times h_{k-1}}$ and *bias vector* $b \in \mathbb{R}^{h_k}$, followed by a non-linear *activation function* $q^{(k)}$, which allows the model to learn general non-linear functions. The output of the last layer represents the final model output or prediction. An illustration of a two-layer MLP is provided in Figure 2.1.

Popular choices for the activation function of the hidden layers of the model include the *sigmoid* function and the *Rectified Linear Unit (ReLU)*. The output layer uses a taskdependent activation function. For classification problems, the *softmax* function

$$y_i = \frac{\exp(z_i)}{\sum_{j=1,\dots,K} \exp(z_j)} \tag{2.3}$$

is commonly used, where the number of output dimensions corresponds to the number of classes, the output dimension with the maximum activation indicates the predicted class, and the outputs provided by softmax are normalized, such that they define a probability distribution over the classes.

The weight matrices and bias vectors of all layers together constitute the model's learnable parameters adjusted during training. Training corresponds to the minimization of a loss function. For classification, a commonly used loss function is the *cross-entropy* loss

$$L(x, \hat{y}) = -\sum_{i=1,\dots,K} y_i \log \hat{y}_i, \tag{2.4}$$

where $\hat{y} \in \{0,1\}^K$ is a one-hot vector indicating the correct label of input x with a one-entry in the corresponding dimension and all remaining entries set to zero. The softmax predictions by the model are given as $y \in \mathbb{R}^K$. Intuitively, the cross-entropy measures the distance between the two probability distributions defined by y and $\hat{y}$, and the goal is to minimize that distance. For a given training dataset, the loss can be averaged over all training samples.

Neural networks are most commonly trained using first-order *gradient descent* optimization methods. After evaluating the loss function for a set of samples, the gradient of the loss function w.r.t. the model parameters is computed and used to update the model parameters using an update rule similar to

$$w_{ij}^{(k)} \leftarrow w_{ij}^{(k)} - \nu \frac{\partial L}{\partial w_{ij}^{(k)}} \tag{2.5}$$

to update parameter $w_{ij}^{(k)}$. Intuitively, the update performs a step in the direction of the negative gradient, i.e., the direction of steepest descent of the loss function, and the step size is determined by the *learning rate* ν. This update step is applied to all model parameters simultaneously and repeated for multiple iterations. This procedure, given appropriate parametrization and a well-defined learning problem, usually converges to at least a local minimum, though theoretical convergence guarantees are usually difficult to establish due to the complex structure and non-convexity of the underlying optimization problems. Different variants of the basic gradient descent algorithm have been introduced in the literature. Arguably, the most commonly used algorithm nowadays is *Adam* [KB15], which additionally maintains individual learning rates for all parameters and adapts these learning rates based on different moments of gradient magnitudes from recent iterations.

The required gradients can be computed using a procedure called *backpropagation*. The main idea is that, due to the sequential structure of stacked neural network layers, the chain rule of differentiation can be used to iteratively compute gradients by propagating back errors from the network output towards the input side. This is illustrated on the right-hand side of Figure 2.1, where the term $\frac{\partial L}{\partial y_i^{(2)}}$ denotes the derivative of the loss function w.r.t. the model output $y_i^{(2)}$, and $\frac{\partial y_i^{(k)}}{\partial z_i^{(k)}}$ corresponds to the derivative of the activation function $q^{(k)}$ w.r.t. $z_i^{(k)}$. Writing $\delta_i^{(k)} = \frac{\partial L}{\partial z_i^{(k)}}$, the derivatives w.r.t. the model parameters can be obtained using, once again, the chain rule:

$$\frac{\partial L}{\partial w_{ij}^{(k)}} = \frac{\partial L}{\partial z_i^{(k)}} \frac{\partial z_i^{(k)}}{\partial w_{ij}^{(k)}} = \delta_i^{(k)} \frac{\partial}{\partial w_{ij}^{(k)}} \left(\sum_{l=1,\ldots,h_{k-1}} w_{il}^{(k)} y_l^{(k-1)} \right) = \delta_i^{(k)} y_j^{(k-1)}. \tag{2.6}$$

That is, the gradient update for weight $w_{ij}^{(k)}$ depends on the backpropagated error $\delta_i^{(k)}$ and the layer input $y_j^{(k-1)}$. It should be noted that much more elaborate models can be trained using this scheme by defining a corresponding computation graph. Differentiation can then be performed automatically, analogous to the simple backpropagation algorithm for MLPs. High-performance libraries offering automatic differentiation on GPUs include the Python libraries *Tensorflow* [ABC+16] and *PyTorch* [PGM+19], which were also employed for most of the prototypical implementations of the algorithms proposed in this thesis.

2.4 Representation Learning

A machine learning algorithm's ability to solve a specific task and the algorithm's effectiveness in doing so depend significantly on the representation of the input data the algorithm is provided with. A common approach is to manually extract a set of features from the raw data and provide these as inputs to the algorithm. For instance, to detect objects in images, one could extract different low-level structures, such as edges, corners, or different types of textures from the image. However, it is usually not clear at all which features are particularly useful for the task, and defining higher-level features, such as specific object parts, is extremely challenging due to sizeable possible variation.

Neural networks offer the possibility of learning representations end-to-end within the model instead of fixing them a-priori. This is also referred to as *representation learning*. Intermediate latent representations learned by neural network layers can be interpreted as learned features of increasing complexity. An illustration of this concept is

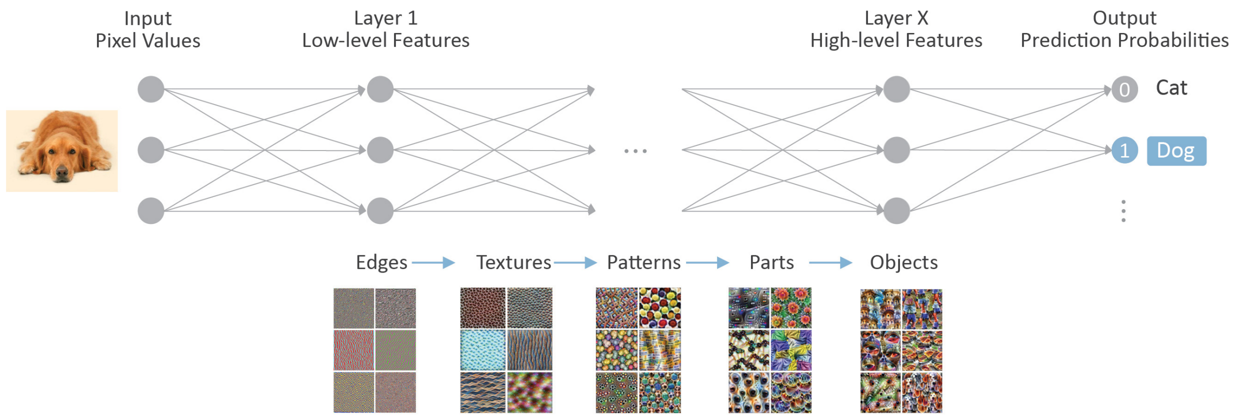


Figure 2.2: Learned hierarchical features in a deep neural network. After low-level features, such as edges and textures, are detected, the model extracts successively higher-level features, including different patterns, object parts, and finally prototypical object representations. For different levels, a selection of features learned by GoogLeNet [SLJ+15] from the ImageNet database [DDS+09] is shown. These visualizations were provided by [OMS17].

provided in Figure 2.2. In early layers, the network extracts low-level representations which form the basis of more high-level representations in the following layers. Learning such representations of increasing complexity with neural networks has been termed in literature as *deep learning* [GBCB16]. Deep learning profits from the flexibility of neural networks combined with simple and robust gradient-based optimization, highly scalable and efficient training and inference on GPUs, and availability of large amounts of data, among other factors, and has celebrated major successes since the early 2010s, significantly pushing benchmark performance on tasks with considerable real-world impact, such as object recognition in images, speech recognition, and machine translation. More recently, deep learning has contributed to enabling technologies such as self-driving vehicles, question-answer systems, and intelligent robots. These advancements have also been enabled by different neural network architectures encoding different inductive biases, including *convolutional neural networks*, which are mainly used on image data, and *recurrent neural networks* and *transformer networks* [VSP+17] for modeling sequential data.

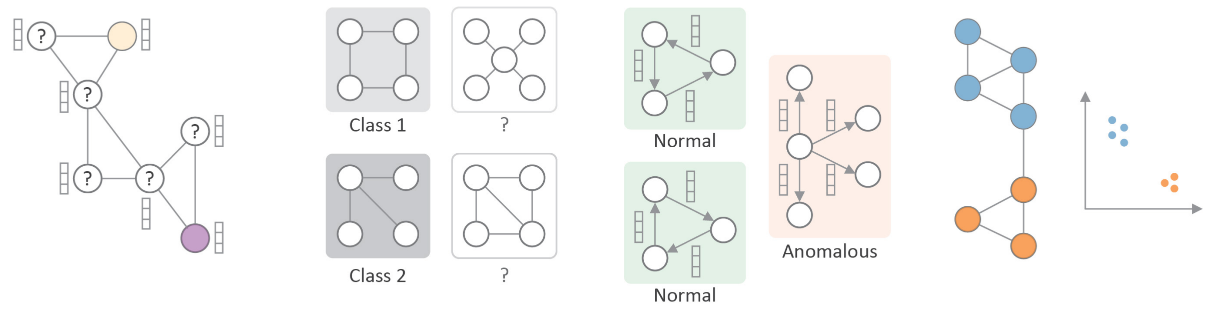
Though deep learning methods continue to significantly advance the state-of-the-art for tasks beyond computer vision and natural language processing, there is still a considerable research effort required to leverage representation learning techniques for other complex types of data. This thesis contributes to this research effort by proposing advanced representation learning algorithms for effective machine learning on graphs and high-dimensional data.

Chapter 3

Machine Learning on Graphs

A graph is a general mathematical formalism that can be used to model connected data. In its most simple form, a graph consists of a set of nodes and a set of edges connecting these nodes, thereby expressing some relationship between the nodes. In many cases, data is naturally organized as a graph. For instance, in a social network, users are connected by friendship or follower edges. Communication networks can model physical information networks or communication between different entities in general, e.g., e-mail correspondence between employees in a company. Interaction networks emerge in various fields and can model different interactions between entities, e.g., interactions between proteins in metabolic processes. Even if a dataset is not explicitly organized as a graph, a graph structure can be extracted from any dataset by connecting samples that are similar in some sense. For instance, if no interactions between specific proteins are known, protein molecules with similar chemical properties could be connected, and their functions predicted based on the resulting graph. This requires a *similarity function* $s : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ where it is typically required that s is symmetric and that each data object is maximally similar to itself.

Formally, a *graph* can be defined as $G = (V, E)$, where V corresponds to the set of *nodes* and E denotes the set of *edges* connecting the nodes. A graph can be directed or undirected. For a *directed graph*, the edge set is given as $E = \{(v_i, v_j) \mid v_i, v_j \in V\}$. For an *undirected graph*, the edges are unordered, i.e., $E = \{\{v_i, v_j\} \mid v_i, v_j \in V\}$. An undirected graph can also be represented as a directed graph with the property that for each edge $(v_i, v_j) \in E$, the backward edge (v_j, v_i) is also contained in E. Algebraically, a graph can be represented as a (usually sparse) *adjacency matrix* $A \in \{0, 1\}^{n \times n}$ with $n = |V|$ denoting the number of nodes and $a_{ij} = 1$ if and only if $(v_i, v_j) \in E$. The adjacency matrix of an undirected graph is symmetric. Graphs can additionally be attributed. Commonly, nodes in the graph can be annotated with feature vectors. These node feature vectors can be collected as rows of a *node feature matrix* $X_V \in \mathbb{R}^{n \times d}$. Similarly, edges can have feature vectors assigned to them. An *edge feature matrix* can be denoted as $X_E \in \mathbb{R}^{m \times d}$



(a) **(b)** **(c)** **(d)**

Figure 3.1: Depictions of different graph learning tasks: (a) Node classification on an undirected graph with node attributes, (b) graph classification on undirected graphs, (c) graph anomaly detection on directed graphs with edge attributes, (d) node embedding on undirected graphs.

where $m = |E|$ denotes the number of edges.

Representing a graph algebraically in terms of an adjacency matrix and optional feature matrices makes it possible to map mathematical operations emerging in graph learning tasks to GPU architectures and perform them jointly with neural network training. Consequently, most graph neural network models can be trained highly efficiently on GPUs.

3.1 Graph Learning Tasks

In this thesis, we consider various machine learning tasks on graphs with all different levels of supervision. The four general learning tasks considered are depicted in Figure 3.1. The following briefly introduces these tasks along with a discussion of state-of-the-art approaches.

3.1.1 Node Classification

The first task we consider is called *semi-supervised node classification* where we are given a single graph G, a node feature matrix X_V and an additional label matrix $Y \in \{0,1\}^{n \times K}$. Labels are provided only for a subset of the nodes, and the goal is to predict the labels of the remaining nodes using the graph structure, the node features, and the labels available for training. The unlabeled nodes are visible during training, making this task semi-supervised. Classification can be binary, multi-class, or multi-label. An example application is document classification, where documents, such as academic publications, are linked by citation or co-authorship links, and each document has a bag-of-words feature vector attached to it describing its content. Given labels for a few documents, the task is to label the remaining documents based on their field of research. Some of the algorithms proposed in this thesis were evaluated on document classification benchmark datasets.

Most existing approaches to semi-supervised node classification assume that close-by nodes in the graph should have a similar label. This assumption is in line with the so-called *homophily* [MSLC01] assumption, which states that actors in social networks usually connect to other actors that share some similarity with them. Such an assumption usually makes sense also for different types of networks.

Earlier semi-supervised node classification methods include *label propagation* methods [ZG02, ZGL03, ZBL$^+$04] where the given labels are propagated through the graph for multiple iterations. Thereby, each node collects the labels from all of its neighbors and aggregates the received labels to update its own label. These steps are repeated until convergence or for a maximum number of iterations. These methods are not able to employ node attributes for classification. Some more recently proposed related methods related to label propagation [SNB$^+$08, NM16, YZM$^+$17] suggest learning more refined classification models for updating node labels based on the observed neighboring labels. By propagating labels to neighbors, label propagation methods assume that close-by nodes in the graph should have a similar label and thus rely on homophily.

Another approach involves explicit *graph-based regularization* [BNS06, WRMC12] where the model's loss function is augmented with a regularization term enforcing that the model produces similar predictions for nodes that are close-by in the graph. These methods are able to incorporate node attributes for classification. A potential downside of this approach is that explicit regularization might impose too strong an inductive bias and thus limit model capacity. This is avoided by another influential method [YCS16] by predicting the graph context for a node in addition to its label instead of imposing explicit regularization. This allows for predicting similar labels even for distant nodes in the graph.

The majority of current state-of-the-art methods belong to the class of *Graph Neural Network* models [BBL$^+$17, GSR$^+$17, BHB$^+$18], which learn expressive node representations by aggregating node attributes among neighborhoods. Recently, these methods have continuously advanced the state-of-the-art in graph learning as a whole, and most of the contributions to graph learning provided in this thesis advance this approach. Graph neural networks are discussed in more detail in Section 3.2.

3.1.2 Graph Classification

In *supervised graph classification*, the task is to classify entire graphs instead of single nodes. The classifier is trained on a set of graphs $\mathcal{G} = \{G_1, \ldots, G_N\}$, optionally corresponding node feature matrices $X_{V1}, \ldots, X_{VN}$, and a label matrix $Y \in \{0,1\}^{N \times K}$. One application that can be modeled as a graph classification task is toxicity prediction, where graphs correspond to substance molecules annotated with chemical properties. The goal is to predict whether a certain substance is toxic or not based on a training set of labeled molecules.

Most existing graph classification approaches can be classified into two major categories, kernel-based methods and feature-based methods.

Kernel-based methods focus on deriving similarity models for pairs of graphs. Thereby, node feature representations are typically learned implicitly, and classification is performed by kernelized classification methods. For instance, the random-walk kernel [BOS$^+$05] starts random walks on both graphs simultaneously and counts the label sequences observed in both graphs. This precedure can be shown to be equivalent to performing a single random walk on the product graph. Further classical graph-kernel methods also count different sub-structures in the graph, e.g., label vectors resulting from the Weisfeiler-Lehman isomorphism test [SSL$^+$11], shortest paths [BK05], or subgraphs [SVP$^+$09] of the two graphs. A problem with these classical graph-kernels is that they are not able to consider correlations between the counted sub-structures. This is addressed by more recent methods [YV15, NCC$^+$16, AZRP18] by learning additional representations of these sub-structures. A disadvantage that remains for kernel-based methods, in general, is that they suffer from relatively high complexity and thus don't scale well. Some other methods focusing on similarity models for graphs directly define metric spaces for graphs [SF83, NMV17, BI18]. The underlying problems being solved by these methods are usually NP-hard and thus require heuristics.

In contrast to kernel-based methods, *feature-based methods* focus on deriving expressive feature representations for graphs that can be used for classification. While some methods [BKERF12] extract hand-crafted features, such as clustering coefficients and centrality scores, to describe a graph, other methods rely on the *SkipGram* model [MCCD13, LM14] to learn a representation of a graph by learning representations of substructures of the graph [NCC$^+$16, AZRP18, LRK18, NLN$^+$18]. Another group of methods [DZHL18, TMK$^+$18] focuses on a graph's spectral properties based on the eigenvalue decomposition of its Laplacian matrix.

All of the above methods extract or learn graph representations or graph similarity models in an unsupervised way, making them somewhat flexible. On the other hand, learning representations end-to-end can improve classification accuracy by learning more task-specific representations. Beyond node classification, graph neural networks can also be employed to solve graph classification tasks by aggregating the learned node representations into a single graph representation used as the input for a prediction layer. While graph neural networks rely on node attributes to learn representations, they can also be applied to unattributed graphs using surrogate node attributes, such as node degree or centrality [NAK16]. In contrast to the above methods, graph neural networks allow for learning graph representations end-to-end and, therefore, for improving classification accuracy.

3.1.3 Graph Anomaly Detection

A further, more general task we consider is *graph anomaly detection*. While different variants of this more general problem exist [ATK15], we focus in the context of this thesis on detecting anomalous graphs in a supervised and an unsupervised setting. The supervised setting corresponds to a binary classification problem where we are given a set of labeled graphs for training, and the goal is to label new graphs as either normal or anomalous. Compared to other binary classification problems, class imbalance is commonly an issue in supervised anomaly detection since the majority of graphs are, by definition, normal. In the unsupervised setting, the training set is unlabeled and consists of normal and anomalous graphs. The model's task is to learn a concept of normality and to classify new graphs as either normal or anomalous. Again, the number of anomalous graphs is assumed to be small compared to the number of normal instances. As an exemplary application, we can consider malware detection, where a communication graph can be extracted from recorded network traffic [BKTS21]. The graph edges are annotated with edge features describing individual communications, e.g., the number of packets sent or the average packet length. The goal is to decide whether the recorded traffic is malicious or benign.

For *supervised anomaly detection*, all graph classification methods discussed in Section 3.1.2 are, in principle, applicable. However, class imbalance should be taken into account during training to ensure that the model learns to discriminate anomalous instances adequately, e.g., by providing a class-balanced sub-sample of the whole dataset to the model for training [HG09]. Similarly, it is important to evaluate anomaly detection results using evaluation measures that adjust for class imbalance [CZS$^+$16], not only in the supervised case.

For *unsupervised anomaly detection*, one approach is to first extract or learn vector space representations from the given graphs in an unsupervised way using methods such as the unsupervised feature-based graph classification methods presented in Section 3.1.2 and to subsequently apply classical anomaly detection methods [CBK09]. Some existing representation learning approaches rely on graph autoencoders to learn graph representations for anomaly detection [Hsu17, GZAL18] instead, but focus on change detection in graph streams. This thesis, on the other hand, contributes two representation learning methods for unsupervised anomaly detection using a graph autoencoder and a graph one-class neural network.

3.1.4 Node Embedding

Another unsupervised problem we consider is the problem of *unsupervised node embedding*. The objective is to learn feature representations of nodes in a single graph in an unsupervised way for down-stream learning tasks, such as node classification or clustering. The learned representations of two nodes should be similar if and only if the nodes are similar w.r.t. the underlying graph structure, e.g., they are close-by or share similar roles in the graph. Such real vector representations are required as input for many general-purpose machine learning algorithms. The learned representations could be used for various supervised and unsupervised tasks, making this approach very flexible. On the other hand, prediction accuracy can usually be improved by learning representations end-to-end, together with the down-stream prediction algorithm.

The majority of existing node embedding methods focus on *homophily*, i.e., nodes are mapped to similar vector space representations if and only if they are close-by in the graph. One pioneering work [PARS14] generates random walks and then learns node representations from the generated walks using a SkipGram model. Thereby, nodes that frequently appear close to each other in a random walk will be mapped to similar representations. Further works have explored different possibilities to bias random walks, e.g., to shift focus between macroscopic and microscopic views [GL16], consider connections between different types of nodes [DCS17], using attention to focus on informative directions for exploration [AEHPARA18], or to avoid problems arising from specific structural graph properties [FBFM18]. Another work focuses on node neighborhoods instead of random walks [TQW$^+$15]. While these methods perform implicit matrix factorization of a node co-occurrence matrix, other methods perform explicit matrix factorization [CLX15]. Further works focus on aspects such as inductivity [HYL17], or uncertainty [BG18].

This thesis contributes a highly-scalable unsupervised algorithm for learning expressive structural node and graph representations. Other graph learning algorithms contributed by this thesis learn node representations in an end-to-end fashion.

3.2 Graph Neural Networks

Graph neural networks (GNNs) [BBL$^+$17, GSR$^+$17, BHB$^+$18] have emerged in recent years as a state-of-the-art approach for learning graph representations for many different graph learning tasks. They provide a powerful and flexible framework for learning node and graph representations and are amenable to highly parallel computation on GPUs [FL19].

While many works are motivated by a spectral graph theory perspective [BZSL14, DBV16, BBL$^+$17, MBM$^+$17], most existing graph neural network algorithms can be expressed within a *neural message passing* framework [DMAI⁺15, LTBZ16, KMB⁺16, KW17, GSR⁺17, HYL17]. The main idea is to propagate node feature vectors through the graph such that each node aggregates feature vectors from its neighborhood. This procedure is formalized in Algorithm 3.1 and visualized in Figure 3.2. Starting with an initial node feature matrix $H^{(0)} = X_V$, node features are propagated to neighboring nodes for a total of K iterations. In each iteration k, each node v_i receives messages $\phi^{(k)}_{j\to i}$ from all of its neighbors v_j, where $\phi^{(k)}$ denotes a *message function* combining both nodes' feature vectors and the weight of the edge from v_j to v_i. Afterwards, v_i aggregates the received messages using an *aggregation function* Aggr and updates its feature vector from the previous iteration using on this aggregate and an update function $\gamma^{(k)}$. All of the above functions, $\phi^{(k)}$, Aggr, and $\gamma^{(k)}$ are required to be differentiable. The aggregation function Aggr is additionally required to be permutation invariant. Note that with every message passing iteration, v_i receives information from one further k-hop neighborhood, such that after K message passing iterations, v_i's feature vector aggregates features from its K-hop neighborhood. A particularly simple and widespread instance of this framework is the *Graph Convolutional Neural Network (GNN)* [KW17], which can be defined by

$$\phi^{(k)}_{j\to i} = \hat{\tilde{\mathrm{A}}}_{ji}\, W^{(k)} h^{(k)}_j \tag{3.1}$$

$$\textsc{Aggr}^{(k)}_i = \sum_{v_j \in \mathcal{N}_i} \phi^{(k)}_{j\to i} \tag{3.2}$$

$$\gamma^{(k)}_i = q^{(k)}\left(\textsc{Aggr}^{(k)}_i\right), \tag{3.3}$$

such that

$$H^{(k+1)} = q^{(k)}\left(\hat{\tilde{\mathrm{A}}}\, H^{(k)} W^{(k)}\right). \tag{3.4}$$

Thereby, $\hat{\tilde{\mathrm{A}}} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ corresponds to a symmetrically normalized version of the adjacency matrix A of G with added self-loops, where $\tilde{A} = A + I$ and $\tilde{D}$ denotes the *degree matrix* of $\tilde{A}$. The message function $\phi^{(k)}$ linearly transforms all feature vectors and weighs each message by the corresponding edge weight $\hat{\tilde{a}}_{ji}$. Incoming messages at each node are summed up and updated using a non-linear activation function $q^{(k)}$, such as ReLU for intermediate layers, or softmax for the final prediction layer. In summary, in each GCN-layer, the node feature vectors are first transformed and then aggregated from direct neighbors as a weighted sum. The symmetric normalization procedure ensures that the scale of the feature vectors does not change after propagation. Self-loops are added to the adjacency matrix to ensure that each node can also consider its own feature

Algorithm 3.1 Synchronous Message Passing

```
Input: Graph $G$, feature matrix $H^{(0)}$
Output: Aggregated feature matrix $H^{(K)}$
  for $k = 1, \ldots, K$ do
      # Send messages
      for $v_i \in V$ do
          for $v_j \in \mathcal{N}_i$ do
              $\phi^{(k)}_{j \to i} = \phi^{(k)}\left(h_i^{(k-1)}, h_j^{(k-1)}, a_{ji}\right)$
          end for
      end for
      # Update node states
      for $v_i \in V$ do
          $h_i^{(k)} = \gamma^{(k)}\left(h_i^{(k-1)}, \textsc{Aggr}_{v_j \in \mathcal{N}_i}\, \phi^{(k)}_{j \to i}\right)$
      end for
  end for
```

vector from the previous iteration during aggregation.

The learned aggregated feature vectors can be used as input for a prediction layer to solve node classification tasks. For graph classification, an additional *pooling* layer can be introduced, which aggregates the learned node representations into a single graph representation used as the input for a prediction layer [HYL17, YYM$^+$18, XHLJ19].

3.2.1 State of the Art

While most works on graph neural networks focus on solving node classification tasks, several works have additionally explored graph classification [NAK16, HYL17, LRK18, YYM$^+$18, XHLJ19] and link prediction [KW16, ZC18] tasks. Another line of research focuses on learning appropriate weights for aggregation instead of just using the weights provided by the (normalized) adjacency matrix. These methods [TWOL18, VCC$^+$18, LRK$^+$19] propose different attention mechanisms to learn such weights based on the two nodes' feature vectors, between which the corresponding message is sent. These attention mechanisms are inspired by the transformer architecture [VSP$^+$17], which introduced the idea of learning attention weights for different parts of a sentence for automatic translation. While graph neural networks perform very well in practice, their actual expressive power is still not very well understood. Several works have explored this [XHLJ19, MRF$^+$19] and proposed different architecture variants improving expressive power in different directions. Despite their effectiveness, most graph neural network models do not scale well to larger graphs. Scalability issues have been addressed, e.g., by sub-sampling neighbors for propagation [CMX18], or by collapsing

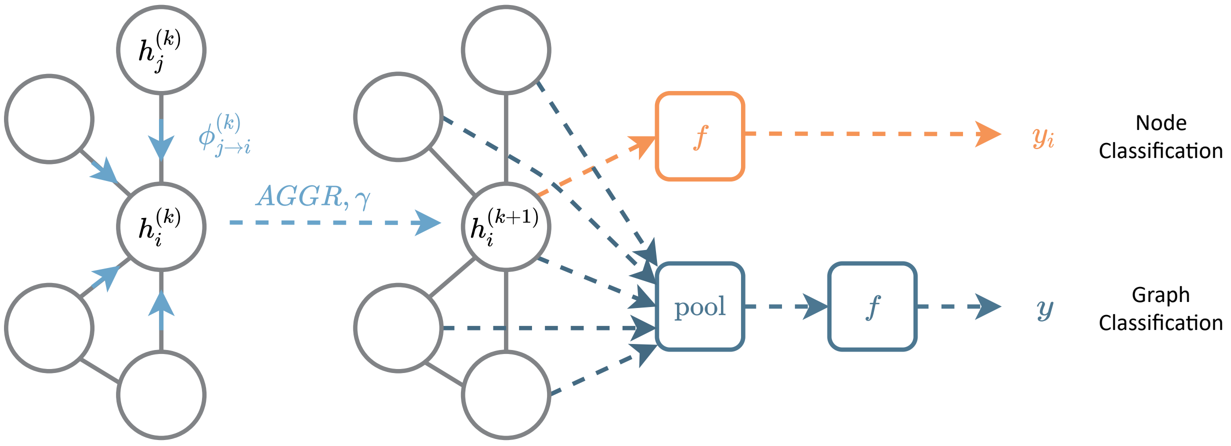


Figure 3.2: During message passing iteration k, node v_i updates its feature vector by first collecting incoming messages from all of its neighbors $v_j \in \mathcal{N}_i$ and subsequently applying the aggregation and update functions. After all K message passing iterations have been performed, node v_i can be classified by feeding its representation $h_i^{(K)}$ to a classifier f, which is typically an MLP. Graph classification can be performed by pooling representations of all nodes in the graph and feeding the pooled representation to a classifier.

multiple message passing layers into a single one by using the K-hop random walk matrix for propagation directly [WSZ+19]. Another approach [KBG19] also relies on the K-hop random walk matrix, but with restarts, resulting in a power iteration approximation of the *Personalized PageRank (PPR)* matrix. For a more detailed presentation and additional discussions on applications, we refer to a recent survey [WPC+20].

3.2.2 Contributions

Our first contribution addresses several shortcomings shared by existing graph neural network models. First, long-range dependencies are not modeled effectively within the synchronous message passing framework since each node sends messages to all neighbors in each iteration. Thus, to send a single message from one node to another distant node, all nodes need to send messages to all other nodes within that distance, even though relatively few long-range dependencies are actually relevant. Apart from efficiency issues, existing works [LHW18, XLT+18] have pointed out an over-smoothing effect, which leads to node feature vectors losing more and more local information as more and more message passing iterations are being performed. Secondly, propagation is restricted to neighbors inside K-hop neighborhoods. Such neighborhoods draw a sharp and somewhat artificial boundary and, therefore, might contain irrelevant neighbors and miss important other nodes. Further, it is necessary to specify the number of message passing iterations, which is somewhat unintuitive.

Our novel graph neural network model *PushNet* [BPS20] addresses these issues by performing asynchronous message passing, instead of synchronous message passing, such that information is pushed through the graph on demand rather than being pulled indiscriminately from all neighbors. We define an asynchronous message passing scheme that we prove to be equivalent to a single synchronous message passing iteration using not the direct neighbors, but neighbors given by *Approximate Personalized PageRank (APPR)* [ABC+07]. Thus, our model is able to perform adaptive asynchronous message passing and, at the same time, allows for highly parallel GPU computation within the synchronous message passing framework.

Additionally, we propose several model variants that allow for flexibly trading off accuracy and efficiency. A multi-scale approach combining propagation results over different neighborhood sizes additionally improves accuracy and allows for simplified hyper-parameter selection. Scalability can be ensured by adapting APPR-related hyper-parameters of our model. In general, the hyper-parameters of our model can be set rather intuitively.

We evaluate our method on five document classification benchmark datasets against state-of-the-art graph neural network models and observe significantly improved classification accuracy. In contrast to competing methods, our model is able to leverage individually adapted neighborhoods for all nodes instead of K-neighborhoods. At the same time, some of our proposed model variants even provide faster runtime.

Our second contribution comprises a novel graph-based approach to network traffic-based malware detection and classification and a corresponding graph neural network model that we name *Network Flow Graph Neural Network (NF-GNN)* [BKTS21]. Malicious software (malware) poses an increasing threat to information technology (IT) and operational technology (OT) systems. Reliable detection and classification of malware is thus a vital task to ensure the safety of such systems. While static approaches analyze a binary executable, dynamic approaches analyze the actual observed behavior of an application. More specifically, we focus on network traffic-based detection, where classification is based on network traffic generated by a candidate application. While there already exist powerful machine learning approaches to this problem [GMP20], none of the existing approaches leverages a graph structure to the best of our knowledge.

Our approach, on the other hand, first extracts an interaction graph from recorded network traffic and subsequently classifies the resulting graph. In contrast to existing approaches that classify individual network flows between two network endpoints during a specific time frame, we collect all traffic within a network during that time frame and represent it by a directed graph with edge attributes. Each edge feature vector contains summary statistics of packets sent. This results in a more holistic view of communication within the network and allows for leveraging intricate communication patterns.

While some existing graph neural network models are able to leverage edge attributes [WPC$^+$20], none of the existing models is directly applicable to our setting.

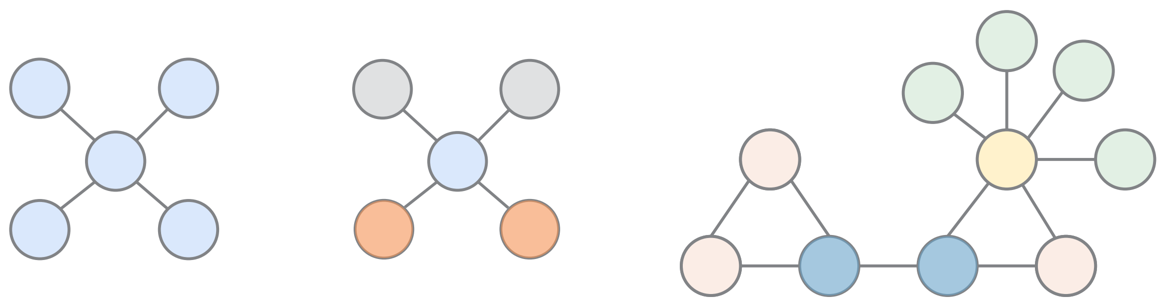
To leverage communication graphs for malware detection and classification, we propose a novel graph neural network model that learns expressive representations of these graphs by iteratively computing node features based on neighboring edges and updating edge features based on incident nodes. Three variants of our model, a classifier, a graph autoencoder, and a graph one-class neural network, are able to solve general supervised and unsupervised graph anomaly detection tasks.

We evaluate our approach empirically on communication graphs extracted from a mobile malware detection benchmark dataset. Each communication graph aggregates network traffic generated during the execution of a single Android application instance during a specific time frame. On supervised binary, malware category, and malware family classification tasks, we improve accuracy over existing approaches by a large margin. Additionally, we demonstrate that our approach provides very accurate detection performance even in an unsupervised setting and settings with small amounts of available data.

3.3 Graph Learning Beyond Homophily

As already briefly addressed above, *homophily* [MSLC01] in graphs refers to the assumption that nodes connected in a graph share some similarity. In social networks, users usually connect to other users, which have similar interests or share other common properties or traits. Such an assumption makes sense in many other settings as well. For instance, nearby sensors in a sensor grid will usually produce similar readings. Similarly, proteins interacting more closely with each other as indicated in a protein interaction network could be expected to share a similar function. For node classification, the homophily assumption states that close-by nodes in the graph should have the same label. Unsupervised node embedding methods are called homophilic if they embed closely-connected nodes to similar embedding vectors.

On the other hand, there are many applications for which either non-homophilic relationships can be expected and a strong homophily assumption is not appropriate, or different kinds of patterns are explicitly sought. Three different kinds of inductive bias considered in this thesis are illustrated in Figure 3.3. For instance, in a social network, users might also interact with other users expressing different opinions, disagreeing with them, or engaging in discussions. Further, we might be explicitly interested in classifying users w.r.t. their structural roles in the network, e.g., influencers or followers. Such roles do not conform to homophily at all, and, in many cases, two connected users will have very different roles.



(a) Homophily **(b)** Heterophily **(c)** Structural similarity

Figure 3.3: Different inductive bias for graph learning: Homophily (a) assumes that neighboring nodes share the same labels, whereas heterophily (b) assumes that nodes connect to each other according to specific label patterns. Structural similarity (c) relates nodes with different structural roles in the graph, e.g., community nodes, bridge nodes, hub nodes, or peripheral nodes.

It is thus vital to consider graph learning methods beyond the homophily assumption. This thesis contributes two such methods.

3.3.1 State of the Art

While most of the graph learning techniques discussed above focus on homophily, many existing approaches allow for or explicitly implement a non-homophilic inductive bias.

Graph neural network models are biased towards homophily by design. This is particularly true for convolutional models [KW17, CMX18, WSZ+19], since they effectively average neighboring feature vectors based on weights provided by the adjacency matrix. Other models, including attention-based models, allow for more flexible edge weights and messages being sent between nodes. Still, they empirically perform poorly on non-homophilic graphs, and developing graph neural networks without a strong homophilic bias is actively researched [ZYZ+20].

In contrast to homophily, *heterophiliy* [Rog10] refers to individuals interacting with other individuals that are dissimilar from themselves w.r.t. certain properties. In graph learning, this corresponds to the assumption that neighboring nodes should have different labels. For instance, users in a social network might engage in discussions with other users that have different opinions on certain topics. In dating networks, the majority of users connect to other users of the opposite gender. While classical label propagation methods focus on homophily, some more recently proposed variants rely on heterophily by propagating labels between nodes with similar adjacency vectors [Pee17], skipping every second node during propagation [Pee17], or reinforcing propagation between nodes with similar labels within their neighborhood [WTT13]. Similarly to label propagation, *belief propagation* methods [Pea82, KKK$^+$11, GKF15] propagate labels between nodes. For that, they rely on transition matrices that explicitly specify relationships between labels, including homophily, heterophily, or mixed patterns, e.g., homophily among one subset of labels and heterophily between another set.

Another relevant non-homophilic inductive bias establishes similarity between nodes based on structural properties. *Structural node embedding* techniques, in contrast to their homophilic counterparts, map nodes to similar representations if and only if they have similar structural properties or fulfill similar structural roles in the graph. For instance, in a social network, one might wish to identify influential users or users bridging different communities. In a network modeling communication within a company, different employees' roles, such as managers or secretaries, could be retraced and analyzed. This can, for instance, be achieved by sampling random walks in a structural similarity graph and learning a SkipGram model [RSF17], or by learning representations from neighborhood degree sequences using a recurrent neural network [TCW$^+$18]. Other works focus more explicitly on role discovery [RA14], mostly based on hand-crafted features [HGER$^+$12]. Diffusion-based approaches to graph classification as discussed in Section 3.1.2 yield structural node representations and also allow for classifying whole graphs based on their structural properties. Another work allows for flexible combinations of homophilic and structural objectives [TMKM18].

3.3.2 Contributions

Our first contribution to this research direction investigates how different patterns among labels, such as homophily, heterophily, or mixed patterns, can be learned by a graph learning model itself instead of relying on fixed assumptions. Our proposed semi-supervised node classification method *Ada-LLD* [FBBS18, FBBS20] learns such patterns by classifying nodes based on the observed distribution of labels in a node's neighborhood and thereby automatically adapts its inductive bias. Consequently, our model is not restricted to fixed patterns, such as those specified a-priori by label or belief propagation methods, but is able to classify nodes based on different learned patterns. In contrast to other models such as graph neural networks, these patterns are learned directly from the observed labels and not inferred indirectly from node attributes.

The neighborhood over which label distributions are computed is determined using APPR. Similarly, as for PushNet, APPR helps avoid irrelevant neighbors inside the K-neighborhood of a node and include relevant ones outside of it. Varying the hyperparameters of APPR makes it possible to compute label distributions over different effective neighborhood sizes and, therefore, to consider patterns at multiple scales. We propose different model variants to combine label distributions over multiple scales for prediction.

Empirically, we evaluate our model on three document classification tasks, a multi-label interest group prediction task in a social network, a multi-label genre prediction task in a movie-actor network, and several synthetic graphs with different predefined label interaction patterns. We observe improved prediction accuracy over other competitors that use only the graph structure for prediction and no further node attributes. Multi-scale combinations of label distributions additionally improve performance.

The second contribution presented in this section allows for learning structural representations of nodes and entire graphs in an unsupervised and particularly simple and scalable fashion. The main idea behind our approach is to define the structural role of a node based on how it assigns importance to its neighbors. For instance, a peripheral node will assign a significant amount of importance to a single node it is connected to, while a hub node will spread importance more evenly among many neighbors. Our method *APPR-Roles* [BBFS17, BBF$^+$19] derives a scalar representation for each node by computing the entropy of its APPR-vector. The entropy function is permutation invariant and intuitively measures the distance from the unit distribution, i.e., the degree of spread of importance among the node's neighbors. The scalar representations can be visualized easily, e.g., by coloring nodes in a drawn graph layout. We represent entire graphs by first clustering all node representations from all graphs in the training dataset using k-Means, where the cluster centroids can be viewed as prototypical roles. The representation of a graph is then obtained by assigning each node representation in the graph to its nearest centroid, counting each centroid's occurrence in the graph, and using the resulting count vector as a representation. Multi-scale role descriptors can be derived by varying parametrization of APPR and concatenating the resulting entropy values.

We evaluate our approach empirically on several real-world datasets. Node classification performance is evaluated on two airport traffic networks. Our model's accuracy closely follows the best competitor while being more than 2300 times faster in terms of computation time. We further evaluate graph classification performance on five biological and five social network datasets. Our model is able to improve accuracy compared to competing methods by up to 10% on five datasets while exhibiting competitive performance on the remaining datasets. Again, our model benefits from fast computation time.

Chapter 4

Machine Learning on High-Dimensional Data

For many machine learning tasks, the task-relevant data is represented by feature vectors, where each data object is described by a d-dimensional numerical vector. Each of the dimensions corresponds to a property or feature, and the value indicates the expression of that feature for the respective data object. For instance, in a medical database, patient data might include age, gender, body height, weight, and different measurements. In an industrial context, a feature vector might describe the state of a machine, where each dimensions corresponds to a sensor and the vector entries contain the sensor readings.

In many cases, data representations are naturally *high-dimensional*. For instance, an image can be represented by a 2-dim. grid of pixels. When flattening the image into a vector, each vector dimension corresponds to a pixel. Even at a relatively small image resolution, the resulting vector will be high-dimensional. Textual documents can be represented as bag-of-words vectors by counting occurrences of each word in a dictionary. Thereby, each dimension corresponds to a word, and the entry counts the number of occurrences in the respective document. Since each document contains only very few words in relation to the size of the whole dictionary, the resulting representations are high-dimensional and sparse. Similarly, user interests can be represented in social networks as binary vectors, where a non-zero entry indicates interest in the corresponding topic. In gene expression analysis, microarray data records expression levels of genes under different conditions or experimental environments. Thereby, for each gene, a large number of expression levels is recorded, resulting in a high-dimensional vector.

4.1 The Curse of Dimensionality

High-dimensional data poses several characteristic challenges to machine learning algorithms. These challenges are related to certain phenomena that arise in high-dimensional spaces and are commonly summarized under the term *curse of dimensionality* [KKZ09]. Compared to lower-dimensional spaces, Euclidean geometry of higher-dimensional spaces exhibits some peculiar and unintuitive effects. This is problematic since machine learning algorithms operating on vector data exploit geometric properties of the data in one way or another. One problem is that norms and distances tend to concentrate. It can be shown that the mean norm of a vector with independent and identically distributed (i.i.d.) dimensions grows logarithmically with the number of dimensions, while the variance stays constant. That is, distances in high-dimensional spaces grow larger and more and more alike and thus become less meaningful. Consequently, the nearest neighbors of a data object tend to be increasingly random and become less meaningful. Another problem is that large regions of a high-dimensional data space are effectively empty. To illustrate this, consider a regular grid spanned over the data space. The number of grid cells will grow exponentially with the number of dimensions, and thus, given a limited amount of data objects, most grid cells will be empty. Similarly, probability densities become more and more heavy-tailed, and relatively low-density regions become increasingly important. Also related to this problem, the number of samples required to estimate a probability density accurately grows exponentially with the number of dimensions. Finally, adding more and more dimensions increases the size of the hypothesis space for a machine learning algorithm and thus makes it prone to over-fitting.

The general problem posed by high-dimensional data to machine learning algorithms can be roughly boiled down to the problem that interesting patterns in the data get obscured by a large number of irrelevant or redundant dimensions. A standard approach to remedy this problem thus consists of eliminating these dimensions. Many feature selection and dimensionality reduction methods have been proposed to reduce the number of dimensions globally. Further, supervised methods commonly employ regularization techniques to counter the increasing size of the hypothesis space [HTF09].

4.2 Subspace Clustering

While global feature selection and dimensionality reduction methods constitute a default approach to dealing with high-dimensional data, for clustering problems, it might not be suitable to reduce the number of dimensions globally since clusters often do not appear in the full-dimensional ambient space but are hidden in lower-dimensional subspaces.

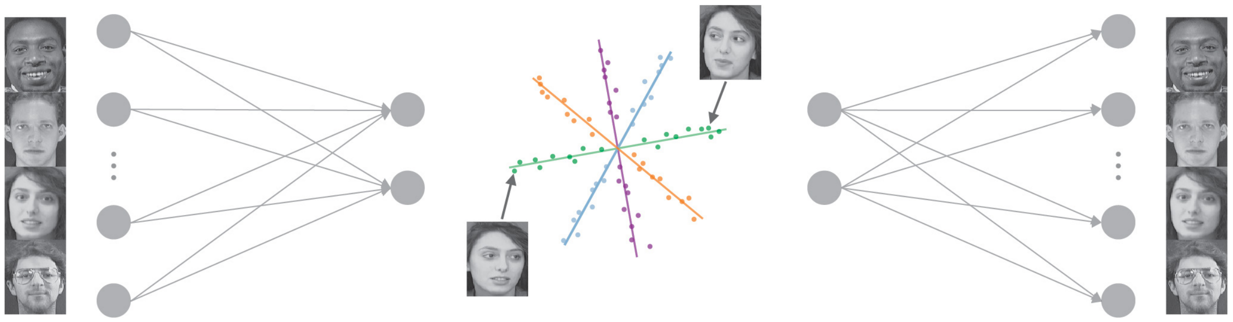


Figure 4.1: Face clustering using a deep subspace clustering model. For each person, a number of images under different conditions, e.g., illumination or pose, are available. In an ideal setting, the images of each person form a low-dimensional subspace, and the subspace dimensions indicate different variations in the images, e.g., change of pose. To account for potential non-linearity, an autoencoder neural network maps the input images into a space that is optimized for linear subspace structure and reconstructs the original images from their latent representations. The face images shown here were taken from the ORL image database [SH94].

For instance, face images of a subject under fixed pose and varying lighting conditions [BJ03], or images of hand-written digits with different rotations, translations, and other natural transformations [HS98] have been shown to lie in low-dimensional subspaces.

For such applications, it makes sense to assume that the dataset was sampled from a collection of lower-dimensional subspaces, rather than the full-dimensional ambient space or a single lower-dimensional subspace. Formally, *subspace clustering* [Vid11] assumes the rows of the data matrix $X \in \mathbb{R}^{n \times d}$ to be sampled from a set of subspaces $\{S_i\}_{i=1}^K$, where each subspace can be written as $S_i = \left\{x \in \mathbb{R}^d \mid \exists y \in \mathbb{R}^{d_i} : x = U_i y\right\}$, where $U_i \in \mathbb{R}^{d \times d_i}$ denotes the basis of subspace S_i and $y \in \mathbb{R}^{d_i}$ is a low-dimensional representation of point $x \in \mathbb{R}^d$. The goal is to correctly assign all points to their respective subspaces and, optionally, to recover the subspace bases. The number of subspaces K and the subspace dimensions d_i are often assumed to be known.

While different variations of the general subspace clustering problem have been investigated in literature [KKZ09, Vid11], we focus on this variant since it is rather general, e.g., it does not restrict subspaces to be axis-aligned, and it is especially amenable to neural network-based representation learning techniques. This thesis provides a novel subspace clustering algorithm that is particularly scalable and comes with a strong theoretical foundation.

4.3 State of the Art

In recent years, methods relying on the so-called *self-expressiveness* property [EV09, LLY10, LLY$^+$12, LMZ$^+$12, EV13, WXL13, FLXY14, JSL14, VF14, JSL15, YLRV16] have established themselves as a state-of-the-art approach to subspace clustering. The main idea behind these methods is that each point can be expressed by a linear combination of other points from the same subspace. Based on this property, an $n \times n$ coefficient matrix is learned, from which cluster labels can be extracted in a subsequent post-processing step using spectral clustering. However, the quadratic number of parameters prevents these methods from scaling beyond smaller datasets. The fact that the number of model parameters depends on the number of data points further makes these models transductive and thus inapplicable to out-of-sample data unseen during training. In contrast, our model requires only a constant number of parameters to provably provide the same expressive power and is inductive, enabling it to cluster out-of-sample data.

In practice, the assumption that the data exactly fits in a set of linear subspaces is often too strong, e.g., due to noise or additional *non-linearity* in the underlying data generating process that has not been accounted for. Since the above methods are not able to appropriately deal with non-linearity, several works propose to rely on *kernels* [CL09, PVNV13, PV14, XTXD15, YGG$^+$16] to learn an implicit transformation to a space, in which the data fits into linear subspaces. However, it is usually not clear whether a particular pre-defined kernel function is particularly for subspace clustering. More recent methods [PXF$^+$16, JZL$^+$17, ZHF18, ZLY$^+$19, KZK20] learn a suitable feature transformation explicitly in an *end-to-end* differentiable model. This idea is illustrated in Figure 4.1. Most notably, *Deep Subspace Clustering Networks (DSC-Net)* [JZL$^+$17] introduced the idea of modeling the coefficient matrix as a dense neural network layer, a so-called self-expressive layer, and training it jointly with an autoencoder. As a result, the encoder represents a feature transformation that has been optimized w.r.t. linear cluster structure in latent space. These methods, however, still rely on a full coefficient matrix and spectral clustering, preventing them from scaling to larger datasets.

While several works have addressed the challenge of *scalability*, some of them are either only able to detect linear clusters [YRV16, RA17]. Others rely on a k-Means-like procedure, which requires good initialization and is sensitive to outliers [ZJH$^+$18], or still fully parametrize coefficient matrices, which need to be re-learned from scratch for each new data batch, and come with no theoretical guarantees [ZJH$^+$19]. In contrast, our model is not only scalable but also suitable to detect non-linear clusters, can be trained end-to-end with back-propagation, and provides a theoretical foundation.

4.4 Contributions

This thesis contributes a novel self-expressiveness-based subspace clustering approach using *Siamese Subspace Clustering Networks (SSCN)* [BFSS20] that is particularly scalable and, at the same time, is provably able to recover the same subspaces as existing self-expressiveness-based subspace clustering methods. Instead of learning a quadratic coefficient matrix directly, our algorithm instead uses a siamese neural network architecture to learn a transformation that maps points into a latent space in which dot products between points correspond to reconstruction coefficients. This constitutes, to the best of our knowledge, the first metric-learning approach to subspace clustering. As a result, our model incurs a quadratic reduction in the number of model parameters. Since the model can be trained with mini-batches, the GPU memory footprint also reduces to a constant size, allowing it to scale to large datasets. An additional classifier trained with self-supervision from the siamese neural network and leveraging some unique geometrical properties of our model further allows for clustering out-of-sample data in an inductive fashion. Finally, to detect non-linear clusters, our model can easily be combined with an autoencoder, which non-linearly maps the input data to a space optimized for linear subspace clustering.

Empirically, we evaluate our model on a hand-written digit image clustering task, where we achieve competitive clustering accuracy compared to existing self-expressiveness-based methods. At the same time, our model can reduce the amount of required GPU memory by multiple orders of magnitude, allowing our model to scale to datasets that have been out of reach of existing algorithms.

Chapter 5

Concluding Remarks

While machine learning has celebrated tremendous success in advancing various fields of research and application in recent years, designing new algorithms that are able to learn effectively from available data remains a challenging feat. Especially for complex data objects, including graphs and high-dimensional data, it is vital to extract expressive features that allow for highly-effective learning. While conventional manual feature engineering methods already enable machine learning algorithms to learn from different types of data, representation learning offers the possibility to automatically learn features that are particularly useful for a task at hand and, therefore, to potentially boost the performance of machine learning algorithms for many different tasks. Especially for complex data, this potential is for from being exhausted yet, and more research work is warranted to fully leverage representation learning techniques for these types of data.

In response to this research demand, this dissertation proposed several novel approaches to learning representations of complex data, in particular, graphs and high-dimensional data, addressing multiple challenges associated with these types of data. These methods allow for highly-effective learning, and most of them are additionally highly scalable, allowing them to learn from massive amounts of data. While we addressed a wide range of general learning problems with different modes of supervision, ranging from unsupervised problems on unlabeled data to (semi-)-supervised learning on annotated data sets, we evaluated our models on specific tasks from fields such as social network analysis, information security, and computer vision.

The first part of this thesis addressed representation learning on graphs. Modeling data as a single or multiple graphs allows machine learning algorithms to leverage relationships present in the data, for instance, to find groups of similar users in a social network for targeted marketing or to predict functional properties of proteins for drug design. Our first contribution advanced graph neural networks by introducing an asynchronous message passing scheme, contrasting existing synchronous message passing methods. Our approach makes message passing more adaptive and efficient, effectively handles long-range dependencies, and can scale even to large graphs. A second contribution addressed the problem of malware detection and classification based on network traffic. We proposed a novel graph-based approach that is able to leverage structurally rich communication graphs in contrast to existing works that classify individual network flows between two endpoints. For detection and classification, we proposed a novel graph neural network model that can be applied to further graph classification or anomaly detection tasks. Two further contributions challenged the homophily assumption commonly made by graph learning methods, which states that nodes with similar properties are usually closely connected in the graph. The first developed method proposed predicting node-level properties based on the distribution of class labels appearing in the respective node's neighborhood. That allows our model to learn general relations between a node and its neighbors, not limited to homophily. Another proposed method specifically models structural similarity between nodes to model different roles, such as influencers and followers in a social network. In particular, we developed an unsupervised algorithm representing nodes based on how they spread probability mass to their neighbors and aggregates node representations to representations for whole graphs.

Future work on representation learning on graphs could explore novel graph neural network models beyond homophily that are able to effectively model different label patterns or structural roles or learn appropriate combinations of different inductive biases. Our asynchronous message passing network framework could be extended by additional methods performing different kinds of push-based message passing or considering temporal graphs. Our proposed malware detection approach could be extended to support better explainability of predictions or explicitly model temporal dynamics. More generally, we plan to further investigate more powerful neighborhood aggregation methods in graph neural networks apart from commonly used simple functions.

The second part of this thesis addressed representation learning on high-dimensional data. More specifically, we considered the problem of clustering high-dimensional data, such as images, texts, or gene expression profiles. Classical clustering algorithms struggle with this type of data since objects are usually not similar w.r.t. to all dimensions of the full-dimensional ambient space but only within a particular subspace of that space. Subspace clustering is an approach to clustering high-dimensional data based on this assumption. While there already exist powerful neural network-based subspace clustering methods, these methods commonly suffer from scalability issues and lack a theoretical foundation. To this end, we proposed a novel metric learning approach to subspace clustering, which can provably recover linear subspaces under suitable assumptions and, at the same time, tremendously reduces the required number of model parameters and memory compared to existing algorithms.

Future work on representation learning on high-dimensional data could explore additional subspace clustering algorithms that are able to automatically learn the number of clusters and subspace dimensions from the data and are still highly effective and scalable. Such methods could also be explored beyond the self-expressiveness property. Further, adapting ideas and concepts from the field of self-supervised learning could be investigated in more detail. While non-linear subspace clustering methods perform very well in practice, it is currently not well understood for which data subspace clustering in latent space is particularly useful, compared to, e.g., centroid-based clustering. In general, it could be investigated which clustering models in latent space should be preferred based on the manifold structure in the ambient space. Beyond clustering, representation learning methods could potentially improve performance on further tasks, such as anomaly detection in high-dimensional data or learning on high-dimensional data streams.

References

[ABC$^+$07] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S Mirrokni, and Shang-Hua Teng. Local computation of pagerank contributions. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 150–165. Springer, 2007.

[ABC$^+$16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[AEHPARA18] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alexander A Alemi. Watch your step: Learning node embeddings via graph attention. *Advances in Neural Information Processing Systems*, 31:9180–9190, 2018.

[ATK15] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688, 2015.

[AZRP18] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. Sub2vec: Feature learning for subgraphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 170–182. Springer, 2018.

[BBF$^+$19] Felix Borutta, Julian Busch, Evgeniy Faerman, Adina Klink, and Matthias Schubert. Structural graph representations based on multiscale local network topologies. In *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, 2019.

[BBFS17] Felix Borutta, Julian Busch, Evgeniy Faerman, and Matthias Schubert. Towards learning structural node embeddings using personalized pagerank. In *Lernen, Wissen, Daten, Analysen (LWDA) Conference Proceedings, Rostock, Germany, September 11-13, 2017*, 2017.

[BBL$^+$17] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[BFSS20] Julian Busch, Evgeniy Faerman, Matthias Schubert, and Thomas Seidl. Learning self-expression metrics for scalable and inductive subspace clustering. *NeurIPS Workshop: Self-Supervised Learning - Theory and Practice*, 2020.

[BG18] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *6th International Conference on Learning Representations (ICLR)*, pages 1–13, 2018.

[BHB$^+$18] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[BI18] Jose Bento and Stratis Ioannidis. A family of tractable graph distances. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 333–341. SIAM, 2018.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[BJ03] Ronen Basri and David W Jacobs. Lambertian reflectance and linear subspaces. *IEEE transactions on pattern analysis and machine intelligence*, 25(2):218–233, 2003.

[BK05] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM'05)*, pages 8–pp, 2005.

[BKERF12] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Netsimile: A scalable approach to size-independent network similarity. *arXiv preprint arXiv:1209.2684*, 2012.

[BKTS21] Julian Busch, Anton Kocheturov, Volker Tresp, and Thomas Seidl. Nf-gnn: Network flow graph neural networks for malware detection and classification. In *33rd International Conference on Scientific and Statistical Database Management (SSDBM)*, 2021.

[BNS06] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7(11), 2006.

[BOS+05] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[BPS20] Julian Busch, Jiaxing Pi, and Thomas Seidl. Pushnet: Efficient and adaptive neural message passing. In *24th European Conference on Artificial Intelligence (ECAI)*, 2020.

[BZSL14] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and deep locally connected networks on graphs. In *2nd International Conference on Learning Representations (ICLR)*, 2014.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[CL09] Guangliang Chen and Gilad Lerman. Spectral curvature clustering (scc). *International Journal of Computer Vision*, 81(3):317–330, 2009.

[CLX15] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900, 2015.

[CMX18] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[CZS+16] Guilherme O Campos, Arthur Zimek, Jörg Sander, Ricardo JGB Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E Houle. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data mining and knowledge discovery*, 30(4):891–927, 2016.

[DBV16] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 3844–3852, 2016.

[DCS17] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 135–144, 2017.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[DMAI+15] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pages 2224–2232, 2015.

[DZHL18] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1320–1329, 2018.

[EV09] Ehsan Elhamifar and René Vidal. Sparse subspace clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2790–2797, 2009.

[EV13] Ehsan Elhamifar and René Vidal. Sparse subspace clustering: Algorithm, theory, and applications. *IEEE transactions on pattern analysis and machine intelligence*, 35(11):2765–2781, 2013.

[FBBS18] Evgeniy Faerman, Felix Borutta, Julian Busch, and Matthias Schubert. Semi-supervised learning on graphs based on local label distributions. In *Proceedings of the 14th International Workshop on Mining and Learning with Graphs (MLG)*, 2018.

[FBBS20] Evgeniy Faerman, Felix Borutta, Julian Busch, and Matthias Schubert. Ada-lld: Adaptive node similarity for node classification using multi-scale local label distributions. *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, 2020. ©2020 IEEE. Reprinted, with permission, from Evgeniy Faerman, Felix Borutta, Julian Busch, and Matthias Schubert, Ada-lld: Adaptive node similarity for node classification using multi-scale local label distributions, IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), 2020.

[FBFM18] Evgeniy Faerman, Felix Borutta, Kimon Fountoulakis, and Michael W Mahoney. Lasagne: Locality and structure aware graph node embedding. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 246–253. IEEE, 2018.

[FL19] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[FLXY14] Jiashi Feng, Zhouchen Lin, Huan Xu, and Shuicheng Yan. Robust subspace segmentation with block-diagonal prior. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3818–3825, 2014.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep Learning*, volume 1. MIT Press Cambridge, 2016.

[GKF15] Wolfgang Gatterbauer Stephan Günnemann, Danai Koutra, and Christos Faloutsos. Linearized and single-pass belief propagation. *Proceedings of the VLDB Endowment*, 8(5), 2015.

[GL16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[GMP20] Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, 2020.

[GSR$^+$17] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272. PMLR, 2017.

[GZAL18] Daniele Grattarola, Daniele Zambon, Cesare Alippi, and Lorenzo Livi. Learning graph embeddings on constant-curvature manifolds for change detection in graph streams. *stat*, 1050:16, 2018.

[HG09] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[HGER$^+$12] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei

Li. Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1231–1239, 2012.

[HS98] Trevor Hastie and Patrice Y Simard. Metrics and models for handwritten character recognition. *Statistical Science*, pages 54–65, 1998.

[Hsu17] Daniel Hsu. Anomaly detection on graph time series. *arXiv preprint arXiv:1708.02975*, 2017.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[HYL17] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[JSL14] Pan Ji, Mathieu Salzmann, and Hongdong Li. Efficient dense subspace clustering. In *IEEE Winter Conference on Applications of Computer Vision*, pages 461–468. IEEE, 2014.

[JSL15] Pan Ji, Mathieu Salzmann, and Hongdong Li. Shape interaction matrix revisited and robustified: Efficient subspace clustering with corrupted and incomplete data. In *Proceedings of the IEEE International Conference on computer Vision*, pages 4687–4695, 2015.

[JZL$^+$17] Pan Ji, Tong Zhang, Hongdong Li, Mathieu Salzmann, and Ian Reid. Deep subspace clustering networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 23–32, 2017.

[KB15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

[KBG19] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *7th International Conference on Learning Representations (ICLR)*, 2019.

[KKK$^+$11] Danai Koutra, Tai-You Ke, U Kang, Duen Horng Polo Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 245–260. Springer, 2011.

[KKZ09] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *Acm transactions on knowledge discovery from data (tkdd)*, 3(1):1–58, 2009.

[KMB$^+$16] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

[KW16] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[KW17] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations (ICLR)*, 2017.

[KZK20] Mohsen Kheirandishfard, Fariba Zohrizadeh, and Farhad Kamangar. Multi-level representation learning for deep subspace clustering. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2039–2048, 2020.

[LHW18] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[LLY10] Guangcan Liu, Zhouchen Lin, and Yong Yu. Robust subspace segmentation by low-rank representation. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 663–670, 2010.

[LLY$^+$12] Guangcan Liu, Zhouchen Lin, Shuicheng Yan, Ju Sun, Yong Yu, and Yi Ma. Robust recovery of subspace structures by low-rank representation. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):171–184, 2012.

[LM14] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[LMZ+12] Can-Yi Lu, Hai Min, Zhong-Qiu Zhao, Lin Zhu, De-Shuang Huang, and Shuicheng Yan. Robust and efficient subspace segmentation via least squares regression. In *European conference on computer vision*, pages 347–360. Springer, 2012.

[LRK18] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. Graph classification using structural attention. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1666–1674, 2018.

[LRK+19] John Boaz Lee, Ryan A Rossi, Sungchul Kim, Nesreen K Ahmed, and Eunyee Koh. Attention models in graphs: A survey. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 13(6):1–25, 2019.

[LTBZ16] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *4th International Conference on Learning Representations (ICLR)*, 2016.

[MBM+17] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5115–5124, 2017.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[MRF+19] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[MSLC01] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.

[NAK16] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023. PMLR, 2016.

[NCC$^+$16] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *Proceedings of the 12th International Workshop on Mining and Learning with Graphs (MLG)*, 2016.

[NLN$^+$18] Dang Nguyen, Wei Luo, Tu Dinh Nguyen, Svetha Venkatesh, and Dinh Phung. Learning graph representation via frequent subgraphs. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 306–314. SIAM, 2018.

[NM16] Sharad Nandanwar and M Narasimha Murty. Structural neighborhood based classification of nodes in a network. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1085–1094, 2016.

[NMV17] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching node embeddings for graph similarity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[OMS17] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[Pea82] Judea Pearl. Reverend bayes on inference engines: a distributed hierarchical approach. In *Proceedings of the Second AAAI Conference on Artificial Intelligence*, pages 133–136, 1982.

[Pee17] Leto Peel. Graph-based semi-supervised learning for relational networks. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 435–443. SIAM, 2017.

[PGM$^+$19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32:8026–8037, 2019.

[PV14] Vishal M Patel and René Vidal. Kernel sparse subspace clustering. In *2014 ieee international conference on image processing (icip)*, pages 2849–2853. IEEE, 2014.

[PVNV13] Vishal M Patel, Hien Van Nguyen, and René Vidal. Latent space sparse subspace clustering. In *Proceedings of the IEEE international conference on computer vision*, pages 225–232, 2013.

[PXF$^+$16] Xi Peng, Shijie Xiao, Jiashi Feng, Wei-Yun Yau, and Zhang Yi. Deep subspace clustering with sparsity prior. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 1925–1931, 2016.

[RA14] Ryan A Rossi and Nesreen K Ahmed. Role discovery in networks. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):1112–1131, 2014.

[RA17] Mostafa Rahmani and George K Atia. Innovation pursuit: A new approach to subspace clustering. *IEEE Transactions on Signal Processing*, 65(23):6276–6291, 2017.

[Rog10] Everett M Rogers. *Diffusion of innovations*. Simon and Schuster, 2010.

[RSF17] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 385–394, 2017.

[SF83] Alberto Sanfeliu and King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (3):353–362, 1983.

[SH94] Ferdinando S Samaria and Andy C Harter. Parameterisation of a stochastic model for human face identification. In *Proceedings of 1994 IEEE workshop on applications of computer vision*, pages 138–142. IEEE, 1994.

[SKB$^+$18] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

[SLJ$^+$15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew

Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[SNB$^+$08] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[SSL$^+$11] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[SVP$^+$09] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pages 488–495, 2009.

[TCW$^+$18] Ke Tu, Peng Cui, Xiao Wang, Philip S Yu, and Wenwu Zhu. Deep recursive network embedding with regular equivalence. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2357–2366, 2018.

[TMK$^+$18] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. Netlsd: hearing the shape of a graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2347–2356, 2018.

[TMKM18] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *Proceedings of the 2018 world wide web conference*, pages 539–548, 2018.

[TQW$^+$15] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

[TWOL18] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

[VCC$^+$18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[VF14] René Vidal and Paolo Favaro. Low rank subspace clustering (lrsc). *Pattern Recognition Letters*, 43:47–61, 2014.

[Vid11] René Vidal. Subspace clustering. *IEEE Signal Processing Magazine*, 28(2):52–68, 2011.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 32:6000–6010, 2017.

[WPC$^+$20] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

[WRMC12] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural networks: Tricks of the trade*, pages 639–655. Springer, 2012.

[WSZ$^+$19] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[WTT13] Bo Wang, Zhuowen Tu, and John K Tsotsos. Dynamic label propagation for semi-supervised multi-class multi-label classification. In *Proceedings of the IEEE international conference on computer vision*, pages 425–432, 2013.

[WXL13] Yu-Xiang Wang, Huan Xu, and Chenlei Leng. Provable subspace clustering: When lrr meets ssc. *Advances in Neural Information Processing Systems*, 26:64–72, 2013.

[XHLJ19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations (ICLR)*, 2019.

[XLT$^+$18] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462. PMLR, 2018.

[XTXD15] Shijie Xiao, Mingkui Tan, Dong Xu, and Zhao Yang Dong. Robust kernel low-rank representation. *IEEE transactions on neural networks and learning systems*, 27(11):2268–2281, 2015.

[YCS16] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[YGG$^+$16] Ming Yin, Yi Guo, Junbin Gao, Zhaoshui He, and Shengli Xie. Kernel sparse subspace clustering on symmetric positive definite manifolds. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5157–5164, 2016.

[YLRV16] Chong You, Chun-Guang Li, Daniel P Robinson, and René Vidal. Oracle based active set algorithm for scalable elastic net subspace clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3928–3937, 2016.

[YRV16] Chong You, Daniel Robinson, and René Vidal. Scalable sparse subspace clustering by orthogonal matching pursuit. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3918–3927, 2016.

[YV15] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015.

[YYM$^+$18] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 4805–4815, 2018.

[YZM$^+$17] Wei Ye, Linfei Zhou, Dominik Mautz, Claudia Plant, and Christian Böhm. Learning from labeled and unlabeled vertices in networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1265–1274, 2017.

[ZBL$^+$04] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. *Advances in neural information processing systems*, 16(16):321–328, 2004.

[ZC18] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 5171–5181, 2018.

[ZG02] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.

[ZGL03] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919, 2003.

[ZHF18] Pan Zhou, Yunqing Hou, and Jiashi Feng. Deep adversarial subspace clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1596–1604, 2018.

[ZJH$^+$18] Tong Zhang, Pan Ji, Mehrtash Harandi, Richard Hartley, and Ian Reid. Scalable deep k-subspace clustering. In *Asian Conference on Computer Vision*, pages 466–481. Springer, 2018.

[ZJH$^+$19] Tong Zhang, Pan Ji, Mehrtash Harandi, Wenbing Huang, and Hongdong Li. Neural collaborative subspace clustering. In *International Conference on Machine Learning*, pages 7384–7393. PMLR, 2019.

[ZLY$^+$19] Junjian Zhang, Chun-Guang Li, Chong You, Xianbiao Qi, Honggang Zhang, Jun Guo, and Zhouchen Lin. Self-supervised convolutional subspace clustering network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5473–5482, 2019.

[ZYZ$^+$20] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

List of Figures

Appendix A

Contributing Publications

Most contributions presented in this dissertation have been invented, evaluated, and published at LMU Munich under the supervision of the author's doctoral advisor Prof. Dr. Thomas Seidl, and in collaboration with colleagues from the Database Systems and Data Mining group. Some contributions have been developed at Siemens Technology, Princeton, NJ, USA, in collaboration with the author's colleagues there, and some have received partial funding from the Munich Center for Machine Learning. All contributions have been previously published. The list of publications, including remarks on the individual authors' contributions, is provided below.

The first contribution on asynchronous message passing for graph neural networks has been published as

[BPS20] Julian Busch, Jiaxing Pi, and Thomas Seidl. Pushnet: Efficient and adaptive neural message passing. In *24th European Conference on Artificial Intelligence (ECAI)*, 2020.

This work was done during an internship at Siemens Technology, Princeton, NJ, USA. The ideas were developed, implemented, and evaluated by the author. The author also prepared the manuscript. Jiaxing Pi and Thomas Seidl contributed to discussions and offered feedback.

The contribution related to network traffic-based malware detection and classification using a novel graph neural network model has been published as

[BKTS21] Julian Busch, Anton Kocheturov, Volker Tresp, and Thomas Seidl. Nf-gnn: Network flow graph neural networks for malware detection and classification. In *33rd International Conference on Scientific and Statistical Database Management (SSDBM)*, 2021.

The article has received a best paper runner-up award. This work was also developed at Siemens Technology, Princeton, NJ, USA, during the same internship. The method was conceptualized, implemented, and evaluated by the author, who also prepared the manuscript. Anton Kocheturov engaged in discussions, provided feedback, assisted experimental evaluation, and helped to prepare the manuscript. Volker Tresp and Thomas Seidl contributed to discussions and offered feedback.

The graph learning approach adapting to general relations between nodes beyond homophily by leveraging local label distributions was first published as a preliminary version. The full version was published later at a different venue:

[FBBS18] Evgeniy Faerman, Felix Borutta, Julian Busch, and Matthias Schubert. Semi-supervised learning on graphs based on local label distributions. In *Proceedings of the 14th International Workshop on Mining and Learning with Graphs (MLG)*, 2018.

[FBBS20] Evgeniy Faerman, Felix Borutta, Julian Busch, and Matthias Schubert. Ada-lld: Adaptive node similarity for node classification using multi-scale local label distributions. *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, 2020. ©2020 IEEE. Reprinted, with permission, from Evgeniy Faerman, Felix Borutta, Julian Busch, and Matthias Schubert, Ada-lld: Adaptive node similarity for node classification using multi-scale local label distributions, IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), 2020.

The full version of the article received the best student paper award. The idea for this approach was developed by Evgeniy Faerman and Felix Borutta. The author contributed to discussions, supported implementation, and experimental evaluation, and contributed to the preparation of the manuscript. Matthias Schubert engaged in discussions and offered feedback.

The unsupervised graph learning method for deriving structural node and graph descriptors was first published as an extended abstract. The full version of the article was published later at a different venue:

[BBFS17] Felix Borutta, Julian Busch, Evgeniy Faerman, and Matthias Schubert. Towards learning structural node embeddings using personalized pagerank. In *Lernen, Wissen, Daten, Analysen (LWDA) Conference Proceedings, Rostock, Germany, September 11-13, 2017*, 2017.

[BBF$^+$19] Felix Borutta, Julian Busch, Evgeniy Faerman, Adina Klink, and Matthias Schubert. Structural graph representations based on multiscale local network topologies. In *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, 2019.

The ideas have been developed by the author, together with Felix Borutta and Evgeniy Faerman. Implementation and evaluation were done by the author and Felix Borutta. Adina Klink implemented and evaluated early prototypes as part of her bachelor's thesis. Matthias Schubert contributed to discussions and provided feedback.

The scalable deep subspace clustering model based on metric learning has been published as

[BFSS20] Julian Busch, Evgeniy Faerman, Matthias Schubert, and Thomas Seidl. Learning self-expression metrics for scalable and inductive subspace clustering. *NeurIPS Workshop: Self-Supervised Learning - Theory and Practice*, 2020.

Conceptualization, implementation, evaluation, and preparation of the manuscript were done by the author. Early prototypes were implemented and evaluated as part of the master's theses of Veronika Sonntag and Tobias Kreuzholz. These theses were supervised by the author and Evgeniy Faerman, who also contributed to discussions. Matthias Schubert and Thomas Seidl engaged in discussions and provided feedback.

24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

PushNet: Efficient and Adaptive Neural Message Passing

Julian Busch [1][2] and **Jiaxing Pi** [1] and **Thomas Seidl** [2]

Abstract. Message passing neural networks have recently evolved into a state-of-the-art approach to representation learning on graphs. Existing methods perform synchronous message passing along all edges in multiple subsequent rounds and consequently suffer from various shortcomings: Propagation schemes are inflexible since they are restricted to k-hop neighborhoods and insensitive to actual demands of information propagation. Further, long-range dependencies cannot be modeled adequately and learned representations are based on correlations of fixed locality. These issues prevent existing methods from reaching their full potential in terms of prediction performance. Instead, we consider a novel asynchronous message passing approach where information is pushed only along the most relevant edges until convergence. Our proposed algorithm can equivalently be formulated as a single synchronous message passing iteration using a suitable neighborhood function, thus sharing the advantages of existing methods while addressing their central issues. The resulting neural network utilizes a node-adaptive receptive field derived from meaningful sparse node neighborhoods. In addition, by learning and combining node representations over differently sized neighborhoods, our model is able to capture correlations on multiple scales. We further propose variants of our base model with different inductive bias. Empirical results are provided for semi-supervised node classification on five real-world datasets following a rigorous evaluation protocol. We find that our models outperform competitors on all datasets in terms of accuracy with statistical significance. In some cases, our models additionally provide faster runtime.

1 Introduction

As a natural abstraction of real-world entities and their relationships, graphs are widely adopted as a tool for modeling machine learning tasks on relational data. Applications are manifold, including documents classification in citation networks, user recommendations in social networks or function prediction of proteins in biological networks.

Remarkable success has been achieved by recent efforts in formulating deep learning models operating on graph-structured domains. Unsupervised node embedding techniques [33, 39, 7, 15, 34] employ matrix factorization to derive distributed vector space representations for further downstream tasks. In settings where labels are provided, semi-supervised models can be trained end-to-end to improve performance for a given task. In particular, graph neural network models [5, 14, 2] have been established as a de-facto standard for semi-supervised learning on graphs. While spectral methods [6, 10, 28, 5] can be derived from a signal processing point of view, a message passing perspective [11, 26, 18, 21, 16, 14] has proved especially useful due to its flexibility and amenability to highly parallel GPU computation [13]. Further recent works have considered additional edge features [14, 36], attention mechanisms [41, 40, 24], addressed scalability [8, 42] and studied the expressive power of graph neural network models [43, 29].

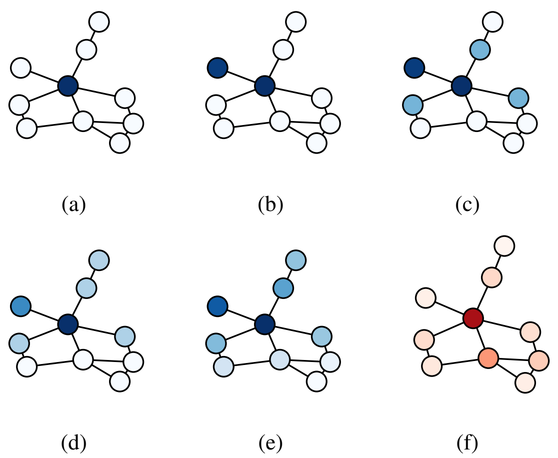


Figure 1: Features of the central node are pushed through the graph until convergence (1a–1e). Equivalently, each node performs a single aggregation step over a node-adaptive neighborhood with importance weights shown for the central node in (1f).

While the above techniques may serve as a basis for modeling further tasks such as link prediction [20, 45] or graph classification [31, 23], we focus on *semi-supervised node classification* in this work. Given a graph $G = (V, E)$, a feature matrix $X \in \mathbb{R}^{n\times d}$ and a label matrix $Y \in \mathbb{R}^{n\times c}$, the goal is to predict labels for a set of unlabeled nodes based on graph topology, node attributes and observed node labels. If no node attributes are available, auxiliary features such as one-hot vectors or node degrees may be used, depending on the task at hand. All graphs considered in the following are undirected, however, extension to directed graphs is straightforward.

Despite their success, existing neural message passing algorithms suffer from several central issues. First, information is pulled indiscriminately from k-hop neighborhoods which will include many irrelevant nodes and miss important ones. In particular, long-range dependencies are modeled ineffectively, since unnecessary messages do not only impede efficiency but additionally introduce noise. Further, interesting correlations might exist on different levels of locality which makes it necessary to consider multi-scale representations. These issues prevent existing neural message passing algorithms from reaching their full potential in terms of prediction per-

[1] Siemens Corporate Technology, Princeton, NJ, USA
jiaxing.pi@siemens.com
[2] Ludwig-Maximilians-Universität München, Munich, Germany
{busch, seidl}@dbs.ifi.lmu.de

Algorithm 1 Synchronous Message Passing

```
Input: Graph $G$, feature matrix $H^{(0)}$
Output: Aggregated feature matrix $H^{(K)}$
  for $k \in [K]$ do
      # Send messages
      for $i \in V$ do
          for $j \in \mathcal{N}_i$ do
              $\phi_{j\to i}^{(k)} = \phi^{(k)}\left(h_i^{(k-1)}, h_j^{(k-1)}, a_{ji}\right)$
          end for
      end for
      # Update node states
      for $i \in V$ do
          $h_i^{(k)} = \gamma^{(k)}\left(h_i^{(k-1)}, \textsc{Aggr}_{j\in\mathcal{N}_i}\, \phi_{j\to i}^{(k)}\right)$
      end for
  end for
```

formance.

To address the above issues, we propose a novel $push$-based neural message passing algorithm which propagates information on demand rather than indiscriminately pulling it from all neighbors. We show that it can be interpreted equivalently as either an asynchronous message passing scheme or a single synchronous message passing iteration over sparse neighborhoods derived from *Approximate Personalized PageRank*. Thereby, each node neighborhood is personalized to its source node, providing a stronger structural bias and resulting in a node-adaptive receptive field. Both views are illustrated in Figure 1. Consequently, our model benefits from the existing synchronous neural message passing framework while providing additional advantages derived from its asynchronous message passing interpretation. In contrast to existing synchronous methods, our model further eliminates the need of stacking multiple message passing layers to reach distant nodes by introducing a suitable neighborhood function. It additionally supports highly efficient training and is able to learn combinations of multi-scale representations.

2 Neural Message Passing Algorithms

Neural message passing algorithms follow a *synchronous* neighborhood aggregation scheme. Starting with an initial feature matrix $H^{(0)} \in \mathbb{R}^{n\times h}$, for K iterations, each node sends a message to each of its neighbors and updates its own state based on the aggregated received messages. Borrowing some notation from [13], we formalize this procedure in Algorithm 1 where $\phi^{(k)}$ is a *message function*, $\textsc{Aggr}$ is a permutation invariant *aggregation function* and $\gamma^{(k)}$ is an *update function*. All of these functions are required to be differentiable.

One of the most simple and widespread representatives of this framework is the *Graph Convolutional Network (GCN)* [21] which can be defined via

$$\phi_{j\to i}^{(k)} = \hat{\tilde{\mathrm{A}}}_{ji}\, W^{(k)} h_j^{(k)} \tag{1}$$

$$\textsc{Aggr}_i^{(k)} = \sum_{j\in\mathcal{N}_i} \phi_{j\to i}^{(k)} \tag{2}$$

$$\gamma_i^{(k)} = q^{(k)}\left(\textsc{Aggr}_i^{(k)}\right) \tag{3}$$

such that

$$H^{(k+1)} = q^{(k)}\left(\hat{\tilde{\mathrm{A}}}\, H^{(k)} W^{(k)}\right) \tag{4}$$

where $H^{(0)} = X$, $q^{(k)}$ is a non-linearity (*ReLU* is used for hidden layers, *softmax* for the final prediction layer), $\hat{\tilde{\mathrm{A}}} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ is a symmetrically normalized adjacency matrix with self-loops and $\tilde{A} = A + I$ with degree matrix $\tilde{D}$. Note that due to self-loops each node also aggregates its own features. Normalization preserves the scale of the feature vectors. In each GCN layer, features are transformed and aggregated from direct neighbors as a weighted sum.

While various models which can be formulated in this framework have achieved remarkable performance, a general issue with synchronous message passing schemes is that long-range dependencies in the graph are not modeled effectively. If $\mathcal{N}_i$ denotes the one-hop neighborhood of node i (which is commonly the case), then each message passing iteration expands the receptive field by one hop. For a single node to gather information from another node of distance K, K message passing iterations need to be performed for *all* nodes in the graph. Sending a large number of unnecessary messages does not only result in unnecessary computation but further introduces noise to the learned node features. On the same note, [44] and [25] pointed out an over-smoothing effect. [44] showed that with an increasing number of layers, node importance in a GCN converges to the graph's random walk limit distribution, i.e., all local information is lost.

2.1 Asynchronous Message Passing

Instead of passing messages along all edges in multiple subsequent rounds, one might consider an *asynchronous* propagation scheme where nodes perform state updates and send messages one after another. In particular, pushing natively supports adaptivity, since instead of just pulling information from all neighbors, nodes are able to push and receive important information on demand. This motivates our *push-based message passing framework* (Algorithm 2).

Algorithm 2 Asynchronous Push-based Message Passing

```
Input: Graph $G$, feature matrix $H^{(0)}$
Output: Aggregated feature matrix $H$
  Initialize $\Phi_i$ for all $i \in V$
  while not converged do
      Select next node $i \in V$
      # Update node state
      $h_i \leftarrow \gamma\left(h_i, \Phi_i\right)$
      # Send messages
      for $j \in \mathcal{N}_i$ do
          $\phi_{i\to j} = \phi\left(h_i, h_j, \Phi_i, \Phi_j, a_{ij}\right)$
          $\Phi_j \leftarrow \textsc{Aggr}\left(\Phi_j, \phi_{i\to j}\right)$
      end for
      reset $\Phi_i$
  end while
```

First, it is important to note that each node needs to aggregate incoming messages until it is selected to be updated. For that purpose, we introduce aggregator states $\Phi_i \in \mathbb{R}^h$ which contain novel unprocessed information for each node. After it is used by a node to update its state and it has pushed messages to its neighbors, the aggregator state is reset until the node receives more information and becomes active again. Further note that the aggregator states naturally lend themselves to serve as a basis for selecting the next node and for a convergence criterion, based on the amount of unprocessed information. The functions ϕ, $\textsc{Aggr}$ and γ fulfill the same roles and share the same requirements as their synchronous counterparts. Though not

Algorithm 3 Local Push Message Passing (LPMP)

```
Input: Graph $G$, feature matrix $H^{(0)}$, parameters $\alpha \in (0,1)$, $\varepsilon > 0$
Output: Aggregated feature matrix $H$
  Initialize dense matrix $H = zeros(n,h)$
  for $k \in V$ do
    $\Phi_i = \delta_{i,k} h_i$ for all $i \in V$
    while $\max_{i \in V} ||\Phi_i|| > \varepsilon ||h_k||$ do
      $h_i \leftarrow h_i + \alpha \Phi_i$
      for $j \in \mathcal{N}_i$ do
        $\Phi_j \leftarrow \Phi_j + \frac{1-\alpha}{d_j} \Phi_i$
      end for
      $\Phi_i = 0$
    end while
  end for
```

specifically indicated, in principle, different functions may be used for different iterations.

As a particular instance of this framework, we propose *Local Push Message Passing (LPMP)* (Algorithm 3). For the next update, the node with the largest aggregator state is selected, since it holds the largest amount of unprocessed information. Similarly, convergence is attained if each node has only a small amount of unprocessed information left. Note that all state updates are additive and no learnable transformations are applied in order to effectively treat long-range dependencies and retain flexibility. Feature transformations may be applied before or after propagation. Further, in each iteration of the outer loop, only the features of node k are diffused through the graph in order to avoid excessive smoothing which might occur when multiple features are propagated at the same time over longer distances in the graph. Also, all iterations of the outer loop are independent of each other and can be performed in parallel. Remaining details of the algorithm will be motivated and explained below.

We further wish to point out that the synchronous framework does not consider any notion of convergence but instead introduces a hyper-parameter for the number of message passing iterations. An early work on *Graph Neural Network (GNN)* [35] applies contraction mappings and performs message passing until node states converge to a fixed point. However, neighborhood aggregation is still performed synchronously.

Finally, further instances of the general push-based message passing framework may be considered in future work. We focus on this particular algorithm due to its nice interpretation in terms of existing push algorithms (as detailed below), its favorable properties and since we observed it to perform well in practice.

3 Pushing Networks

The LPMP algorithm described above is inspired by local *push* algorithms for computation of *Approximate Personalized PageRank (APPR)* [17, 3] and, in particular, we will show in the following how it can be equivalently described as a single synchronous message passing iteration using sparse APPR neighborhoods. Thus, the proposed message passing scheme effectively combines the advantages of existing synchronous algorithms with the benefits of asynchronous message passing described above.

3.1 Personalized Node Neighborhoods

Personalized PageRank (PPR) refers to a localized variant of *PageRank* [32] where random walks are restarted only from a certain set of nodes. We consider the special case in which the starting distribution is a unit vector, i.e., when computing PPR-vector π_i of node i, walks are always restarted at i itself. Formally, π_i can be defined as the solution of the linear system

$$\pi_i = \alpha e_i + (1-\alpha)\pi_i A_{rw} \tag{5}$$

where $e_i \in \mathbb{R}^n$ denotes the ith unit vector, $A_{rw} = D^{-1}A$ denotes the random walk transition matrix of G, and the restart probability $\alpha \in (0,1)$ controls the locality, where a larger value leads to stronger localization. The PPR vectors for all nodes can be stored as rows of a PPR-matrix $\Pi \in \mathbb{R}^{n \times n}$. Intuitively, π_{ij} corresponds to the probability that a random walk starting at i stops at j where the expected length of the walk is controlled by α. The vector π_i can be interpreted as an importance measure for node i over all other nodes where π_{ij} measures the importance of j for i. Since these measures are not sparse and global computation of Π would require $\mathcal{O}(n^2)$ operations, we consider local computation of APPR instead. In particular, we refer to the *Reverse Local Push* algorithm [1], since it comes with several useful theoretical properties. Complexity of computing the whole matrix P is reduced to $\mathcal{O}(n/\alpha\varepsilon)$ [1], i.e., linear in the number of nodes. The parameter $\varepsilon > 0$ controls the quality of approximation, sparsification and runtime where a larger value leads to sparser solutions. For a more in-depth discussion, we refer to [1].

3.2 PushNet

Based on the above neighborhood function, we propose the following neural message passing algorithm:

Definition 1 (PushNet) *Let f and g be MLPs parametrized by θ_f and θ_g, respectively, h_1, h_2, h_3 denote hidden dimensions, $P = \left[P^{(\alpha_1)}, \dots, P^{(\alpha_K)}\right] \in \mathbb{R}^{K \times n \times n}$ be a tensor storing precomputed APPR matrices for different scales $\alpha_1 \geq \cdots \geq \alpha_K$ and* AGGR *denote a differentiable* scale aggregation function*. Given input features $X \in \mathbb{R}^{n \times d}$, the layers of* PushNet *are defined as*

$$H^{(0)} = f(X; \theta_f) \qquad \in \mathbb{R}^{n \times h_1} \tag{6}$$

$$H^{(1)} = PH^{(0)} \qquad \in \mathbb{R}^{K \times n \times h_1} \tag{7}$$

$$H^{(2)} = \text{AGGR}\left(H^{(1)}\right) \qquad \in \mathbb{R}^{n \times h_2} \tag{8}$$

$$H^{(3)} = g\left(H^{(2)}; \theta_g\right) \qquad \in \mathbb{R}^{n \times h_3} \tag{9}$$

In most cases, $h_3 = c$, such that $H^{(3)}$ provides the final predictions for each node over c classes. In general, PushNet might also be applied to different graph learning problems such as graph classification, where learned node representations are pooled and labels are predicted for whole graphs. However, we leave these further applications to future work.

To draw the connection between synchronous and asynchronous message passing, we show that the base variant of PushNet with no feature transformations and a single scale α is equivalent to LPMP (Algorithm 3):

Theorem 1 *Let $\alpha \in (0,1)$ and $\varepsilon > 0$ be fixed, $f, g,$* AGGR *be identity functions and $K = 1$. Then $H^{(3)} = H$ where $H^{(3)} = PX$ is the output of PushNet and H is the output of LPMP.*

The main idea is that instead of propagating features directly as in LPMP, we can first propagate scalar importance weights as in Reverse Local Push and then propagate features in a seconds step. Thus, all discussion on LPMP are directly applicable to PushNet, including adaptivity, effective treatment of long-range dependencies and avoidance of over-smoothing. We wish to point out that an additional interpretation of adaptivity can be derived from the perspective of PushNet: APPR-induced neighborhoods of different nodes are sparse and directly exclude irrelevant nodes from consideration, in contrast to commonly used k-hop neighborhoods. In this sense, APPR is adaptive to the particular source node. To the best of our knowledge, no existing neural message passing algorithm shares this property.

In practice, it is favorable to not propagate features using LPMP, but to pre-compute APPR matrices such that features are propagated only once along all non-zero APPR entries and there is no need to propagate gradients back over long paths of messages. Thus, PushNet benefits from the existing synchronous neural message passing framework while providing additional advantages derived from its asynchronous interpretation.

3.3 Learning Multi-Scale Representations

Additional properties of PushNet compared to LPMP include feature transformations f and g which may be applied before and after feature propagation. Since the optimal neighborhood size cannot be assumed to be the same for each node and patterns might be observed at multiple scales, we additionally propagate features over different localities by varying the restart probability α. The multi-scale representations are then aggregated per node into a single vector such that the model learns to combine different scales for a given node. In particular, we consider the following scale aggregation functions:

- **sum**: Summation of multi-scale representations. Intuitively, sum-aggregation corresponds to an unnormalized average with uniform weights attached to all scales.

$$\text{AGGR}\left(H^{(1)}\right) = \sum_{k\in[K]} P^{(\alpha_k)} H^{(0)} \in \mathbb{R}^{n\times h_1} \tag{10}$$

  Note that due to distributivity, PushNet with sum aggregation reduces to propagation with a single matrix $P = \sum_{k\in[K]} P^{(\alpha_k)}$, i.e., features can be propagated and additively combined over an arbitrary number of different scales at the cost of only a single propagation. Thereby, the non-zero entries in P are given by $\text{nz}\,(P) = \bigcup_{k\in[K]} \text{nz}\left(P^{(\alpha_k)}\right)$. However, usually the number of non-zero entries $\text{nnz}\,(P)$ will be close to $\text{nnz}\left(P^{(\alpha_K)}\right)$, since nodes considered at a smaller scale will most often also be considered at a larger scale. Thus, complexity will be dominated by the largest scale considered.
- **max**: Element-wise maximum of multi-scale representations. The most informative scale is selected for each feature individually. This way, different features may correspond to more local or more global properties.

$$\text{AGGR}\left(H^{(1)}\right) = \max_{k\in[K]} P^{(\alpha_k)} H^{(0)} \in \mathbb{R}^{n\times h_1} \tag{11}$$

- **cat**: Concatenation of multi-scale representations. Scale combination is learned in subsequent layers. The implied objective is to learn a scale aggregation function which is globally optimal for all nodes.

$$\text{AGGR}\left(H^{(1)}\right) = \Big\|_{k\in[K]} P^{(\alpha_k)} H^{(0)} \in \mathbb{R}^{n\times (K\cdot h_1)} \tag{12}$$

3.4 The PushNet Model Family

We wish to point out several interesting special cases of our model. In our default setting, prediction layers will always be dense with a *softmax* activation. If hidden layers are used, we use a single dense layer with *ReLU* activation.

- **PushNet.** The general case in which f and g are generic MLPs. As per default, f is a single dense hidden layer and g is a dense prediction layer.
- $\mathbf{f = id}$. No feature transformation is performed prior to propagation. In this case, $H^{(2)}$ needs to be computed only once and can then be cached, making learning extremely efficient. The following sub-cases are of particular interest:
  - **PushNet-PTP.** The sequence of operations is "push – transform – predict". In this case, g is a generic MLP, consisting of 2 layers per default.
  - **PushNet-PP.** The model performs operations "push – predict" and uses no hidden layers. Predictions can be interpreted as the result of logistic regression on aggregated features. This version is similar to SGC [42] with the difference that SGC does not consider multiple scales and propagates over k-hop neighborhoods.
  - **LPMP.** In this setting, $g = id$ and $K = 1$, i.e., no feature transformations are performed and features are aggregated over a single scale. This setting corresponds to LPMP, cf. Theorem 1. Note that this model describes only feature propagation, no actual predictions are made.
- **PushNet-TPP.** The setting is $h_1 = c$ and $g = id$, such that the model first predicts class labels for each node and then propagates the predicted class labels. This setting is similar to APPNP [22] but with some important differences. APPNP considers only a single fixed scale and does not propagate over APPR neighborhoods. Instead, multiple message passing layers are stacked to perform a power iteration approximation of PPR. The resulting receptive field is restricted to k-hop neighborhoods. Note that *cat* aggregation is not applicable here.

3.5 Comparison with Existing Neural Message Passing Algorithms

Existing message passing algorithms have explored different concepts of node importance, i.e., weights used in neighborhood aggregation. While GCN [21] and other GCN-like models use normalized adjacency matrix entries in each layer, *Simplified Graph Convolution (SGC)* [42] aggregates nodes over k-hop neighborhoods in a single iteration using a k-step random walk matrix. *Approximate Personalized Propagation of Neural Predictions (APPNP)* [22] also relies on a k-step random walk matrix but with restarts which can be equivalently interpreted as a power iteration approximation of the PPR matrix. *Graph Attention Network (GAT)* [41] learns a similarity function which computes a pairwise importance score given two nodes' feature vectors. All of the above methods aggregate features over fixed k-hop neighborhoods. PushNet on the other hand aggregates over sparse APPR neighborhoods using the corresponding importance scores.

Multi-scale representations have been considered in *Jumping Knowledge Networks (JK)* [44] where intermediate representations of a GCN or GAT base network are combined before prediction. The original intention was to avoid over-smoothing by introducing these

	$\|\mathbf{V}\|$	$\|\mathbf{E}\|$	$\mathbf{d}$	$\mathbf{c}$	$\mathbf{avgSP}$	$\mathbf{maxSP}$
CiteSeer	2120	3679	3703	6	9.33	28
Cora	2485	5069	1433	7	6.31	19
PubMed	19717	44324	500	3	6.34	18
Coauthor CS	18333	81894	6805	15	5.43	24
Coauthor Physics	34493	247962	8415	5	5.16	17

Table 1: Dataset statistics including average and maximum shortest path lengths considering only the largest connected component of each graph.

skip connections. Similarly, [27] concatenate propagated features at multiple selected scales. LD [12] uses APPR to compute local class label distributions at multiple scales and proposes different combinations but is limited to unattributed graphs. PushNet varies the restart probability in APPR to compute multi-scale representations and combines them using simple and very efficient aggregation functions. APPR-Roles [4] also employs APPR but to compute structural node embeddings in an unsupervised setting. The idea of performing no learnable feature transformations prior to propagation was explored already in SGC [42]. It allows for caching propagated features, resulting in very efficient training, and is also used by PushNet-PTP and PushNet-PP. LD [12] explored the idea of propagating class labels instead of latent representations. Similarly, APPNP [22] and PushNet-TPP propagate predicted class labels.

To the best of our knowledge, the only existing work considering asynchronous neural message passing is SSE [9]. Compared to PushNet, SSE is pull-based, i.e., in each iteration a node pulls features from all neighbors and updates its state, until convergence to steady node states. To make learning feasible, stochastic training is necessary. Further, the work focuses on learning graph algorithms for different tasks and results for semi-supervised node classification were not very competitive. In contrast, PushNet offers very fast deterministic training and adaptive state updates due to a push-based approach.

4 Experiments

We compare PushNet and its variants against six state-of-the-art models, GCN [21], GAT [41], JK [44] with base model GCN and GAT, SGC [42], *Graph Isomorphism Network (GIN)* [43] and APPNP [22] on five established node classification benchmark datasets. For better comparability, all models were implemented using *PyTorch Geometric* [13] [3] and trained on a single NVIDIA GeForce GTX 1080 Ti GPU.

4.1 Datasets

Experiments are performed on semi-supervised text classification benchmarks. In particular, we consider three citation networks, *CiteSeer* and *Cora* from [37] and *PubMed* from [30], and two co-authorship networks, *Coauthor CS* and *Coauthor Physics* from [38]. Statistics of these datasets are summarized in Table 1.

4.2 Experimental Setup

For the sake of an unbiased and fair comparison, we follow a rigorous evaluation protocol, similarly as in [38] and [22]. [4] We restrict all graphs to their largest connected components and $L1$-normalize all feature vectors. Self-loops are added and different normalizations are applied to the adjacency matrices individually for each method as proposed by the respective authors. For each dataset, we sample 20 nodes per class for training and 500 nodes for validation. The remaining nodes are used as test data. Each model is evaluated on 20 random data splits with 5 random initializations, resulting in 100 runs per model and dataset. Using the same random seed for all models ensures that all models are evaluated on the same splits.

[3] https://github.com/rusty1s/pytorch_geometric
[4] Note that results for competing methods might differ from those reported in related work due to a different experimental setup.

	Hidden size	**Learning rate**	**Dropout**	L_2 **reg. strength**
GCN	64	0.01	0.5	0.001
GAT	64	0.01	0.6	0.001
SGC	—	0.01	—	0.0001
GIN	64	0.01	0.6	—
JK-GCN (cat)	64	0.01	0.4	0.001
JK-GAT (cat)	64	0.01	0.4	0.001
APPNP ($\alpha = 0.1, K = 10$)	64	0.01	0.5	0.01
PushNet (sum)	64	0.005	0.5	0.01
PushNet-PTP (sum)	64	0.005	0.3	0.1
PushNet-PP (sum)	—	0.01	0.6	0.001
PushNet-TPP (sum)	32	0.01	0.5	0.01

Table 2: Optimal hyper-parameters for all models as determined by a grid search on CiteSeer and Cora. Values in parentheses indicate optimal model-specific hyper-parameters.

Model architectures including sequences and types of layers, activation functions, locations of dropout and $L2$-regularization are fixed as recommended by the respective authors. All remaining hyperparameters are optimized per model by selecting the parameter combination with best average accuracy on CiteSeer and Cora validation sets. Final results are reported only for the test sets using optimal parameters. All models are trained with *Adam* [19] using default parameters and early stopping based on validation accuracy and loss as in [41] with a patience of 100 for a maximum of 10000 epochs.

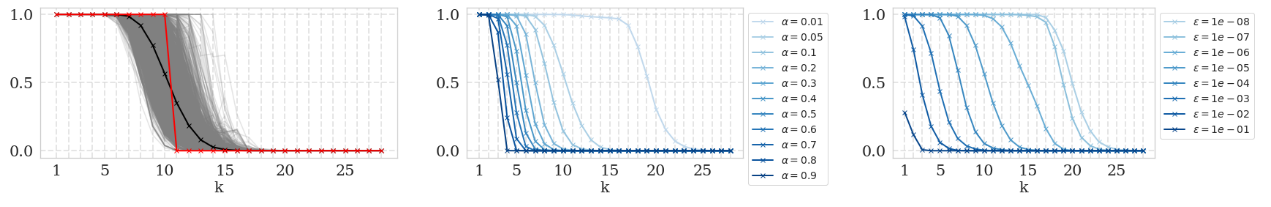
For all PushNet variants, we fix the architecture as described in the previous section. Dropout is applied to all APPR matrices and to the inputs of all dense layers. However, for PushNet-PTP and PushNet-PP dropout is only applied after propagation, such that propagated features can be cached. $L2$-regularization is applied to all dense layers. As a default setting, we consider three different scales $\alpha \in \{0.2, 0.1, 0.05\}$ and $\varepsilon = 1e-5$. Due to memory constraints, we use $\varepsilon = 1e-4$ on Physics dataset for all PushNet variants and on CS and PubMed datasets for PushNet and PushNet-TPP. We further add self-loops to the adjacency matrices and perform symmetric normalization as in GCN. All APPR-matrices are $L1$-normalized per row. We use the following parameter grid for tuning hyper-parameters of all models:

- Number of hidden dimensions: [8, 16, 32, 64]
- Learning rate: [0.001, 0.005, 0.01]
- Dropout probability: [0.3, 0.4, 0.5, 0.6]
- Strength of L2-regularization: [1e-4, 1e-3, 1e-2, 1e-1]

Except for APPNP, all competitors use $K = 2$ propagation layers. JK and GIN use an additional dense layer for prediction. For GAT layers, the number of attention heads is fixed to 8. Optimal hyper-parameters for all models are reported in Table 2.

4.3 Node Classification Accuracy

Accuracy/micro-F1 scores for all datasets are provided in Table 3. It can be observed that our models consistently provide best results on



(a) CiteSeer ($\alpha = 0.05$, $\varepsilon = 1e-5$) (b) CiteSeer ($\varepsilon = 1e-5$) (c) CiteSeer ($\alpha = 0.05$)

Figure 2: Fraction of k-hop neighbors contained in APPR-neighborhoods for CiteSeer. (2a) shows all APPR neighborhoods for a fixed setting, including mean and the closest k-neighborhood for reference. (2b) and (2c) demonstrate localization depending on α and sparsification depending on ε, respectively.

	GCN	**GAT**	**JK-GCN**	**JK-GAT**	**SGC**	**GIN**	**APPNP**	**PushNet**	**PushNet-PTP**	**PushNet-PP**	**PushNet-TPP**
CiteSeer	72.82 ± 1.48	73.82 ± 1.35	71.09 ± 1.66	71.76 ± 1.27	73.91 ± 1.30	70.81 ± 1.61	74.36 ± 1.44	75.08 ± 0.99	$\mathbf{75.19 \pm 1.15}$	75.17 ± 1.32	75.01 ± 1.11
Cora	81.07 ± 1.43	82.12 ± 1.41	79.57 ± 1.63	80.10 ± 1.52	80.13 ± 2.15	80.24 ± 1.54	83.58 ± 1.03	84.12 ± 1.08	83.41 ± 1.24	81.52 ± 1.40	$\mathbf{84.23 \pm 1.26}$
PubMed	78.29 ± 1.48	78.21 ± 1.60	77.23 ± 2.01	77.59 ± 2.25	77.00 ± 1.78	77.19 ± 1.75	79.61 ± 2.98	79.80 ± 1.39	$\mathbf{80.22 \pm 1.27}$	77.52 ± 2.05	80.10 ± 1.33
Coauthor CS	91.64 ± 0.62	90.20 ± 0.75	91.60 ± 0.54	92.20 ± 0.43	91.27 ± 0.58	91.46 ± 0.54	91.10 ± 1.12	92.40 ± 0.52	92.37 ± 0.40	91.04 ± 0.76	$\mathbf{92.54 \pm 0.34}$
Coauthor Physics	93.42 ± 0.63	93.43 ± 0.50	93.49 ± 0.56	*o.o.m.*	*o.o.m.*	93.79 ± 0.49	93.96 ± 0.45	94.01 ± 0.53	93.97 ± 0.48	93.67 ± 0.55	$\mathbf{94.09 \pm 0.47}$

Table 3: Accuracy/micro-F1 scores on semi-supervised node classification datasets in terms of mean and standard deviation over 100 independent runs. JK-GAT and SGC are out of GPU memory on the largest dataset, Coauthor Physics.

	CiteSeer	**Cora**	**PubMed**	**Coauthor CS**	**Coauthor Physics**
Accuracy	4.18e-08	3.37e-07	1.20e-02	4.62e-11	3.45e-03
Macro-F1	3.91e-10	2.79e-04	2.22e-02	9.22e-12	2.12e-07

Table 4: P-values according to a Wilcoxon signed-rank test comparing the best of our models with the best competitor on all datasets with respect to accuracy/micro-F1 and macro-F1.

all datasets and that the strongest model, PushNet-TPP, outperforms all competitors on all datasets. Improvements of our best model compared to the best competing model are statistically significant with $P < .05$ on all datasets according to a Wilcoxon signed-rank test. [5] P-values for all datasets are reported in Table 4. On CiteSeer and PubMed, PushNet-PTP is able to push performance even further. PushNet with feature transformations before and after propagation is less performant but still outperforms all competitors on all datasets. PushNet-PP, the most simple of our models, performs worst as expected. However, it is still competitive, outperforming all competitors on CiteSeer. Boxplots shown in Figure 3 indicate that our models generally exhibit small variance and are less prone to produce outlier scores.

We argue that improvements over existing methods are primarily due to push-based propagation. Figure 2a compares APPR neighborhoods with k-hop neighborhoods in terms of the fraction of k-neighbors considered. It can be seen that k-neighborhoods used by competitors draw a sharp artificial boundary while APPR adaptively selects nodes from larger neighborhoods and discards nodes from smaller ones, *individually for each source node*. Visually, deviations left to the boundary correspond to discarded irrelevant nodes, while deviations on the right hand side indicate additional nodes beyond the receptive field of competitors that can be leveraged by our method. Stacking more message passing layers to reach these nodes would degrade performance due to overfitting as demonstrated in [44] and [25].

Among the competing methods, APPNP performs best in general, providing best baseline performance on all datasets but CS where JK-GAT achieves best results. GAT outperforms GCN on three datasets, CiteSeer, Cora and Physics. JK performs worse than its respective basemodel in most cases. Similar observations were already made in [22]. SGC mostly performs worse than GCN due to its simplicity, outperforming it only on CiteSeer. GIN also provides worse results than GCN in most cases, possibly due to overfitting caused by larger model complexity. It outperforms GCN only on Physics, providing results similar to APPNP.

On CiteSeer, models using cached features perform very well, even the simple models SGC and PushNet-PP which effectively perform linear regression on propagated raw features provide superior performance. On the remaining datasets, the additional feature transformation provided by PushNet-PTP is necessary to guarantee high accuracy.

Macro-F1 scores reveal similar insights and are ommitted due to space constraints.

4.4 Runtime

Figure 4 compares all methods based on average runtime per epoch and accuracy. [6] SGC has lowest runtime on all datasets but runs out of memory on Physics since it propagates raw features over k-hop neighborhoods. PushNet-PP performs second fastest, followed by PushNet-PTP which generally provides a good tradeoff between runtime and accuracy. PushNet and PushNet-TPP are slower than competitors but still provide comparable runtime at a superior level of accuracy. Among the competitors, APPNP, GAT and JK-GAT require most computation time. JK-GAT also runs out of memory on Physics.

[5] In fact, results are significant with $P < 0.01$ on all datasets except PubMed.

[6] We note that for APPNP and all PushNet variants, (A)PPR matrix computation is not included in runtime analysis such that runtime comparison is solely based on propagation, transformation and prediction for all compared models. We consider (A)PPR computation as a preprocessing step, since it is only required to be performed once per graph and can then be reused by all models for this graph. Computation is very fast for each of the considered graphs and can be performed on CPU.

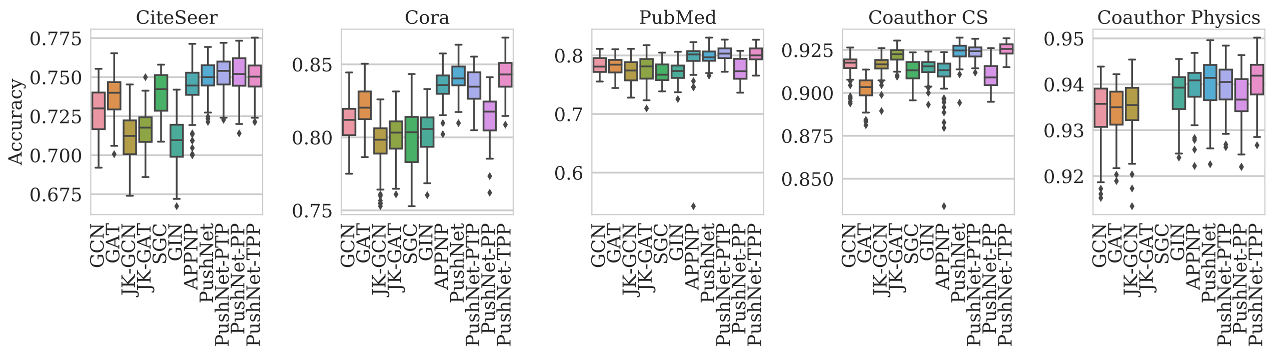


Figure 3: Accuracy/micro-F1 scores on semi-supervised node classification datasets aggregated over 100 independent runs.

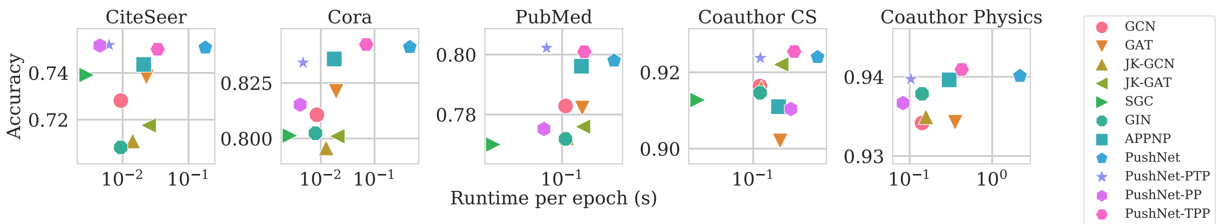


Figure 4: Comparison w.r.t. average runtime per epoch in seconds vs. average accuracy.

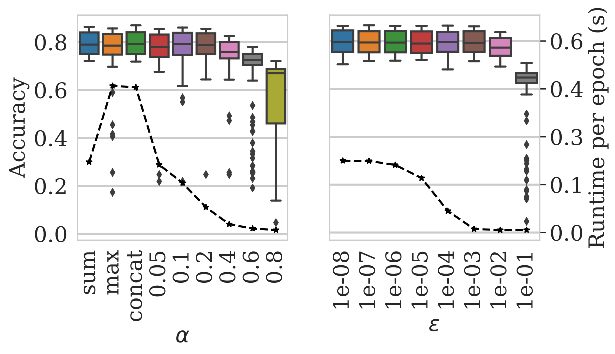


Figure 5: Effect of varying α for fixed $\varepsilon = 1e-5$ (left) and varying ε for fixed $\alpha \in \{0.05, 0.1, 0.2\}$ with *sum* aggregation (right). Dashed lines indicate average runtime per epoch in seconds.

4.5 Influence of Locality

To study the influence of the locality parameter α on the performance of our models, we run experiments with our base model PushNet with various single α values and different aggregations of multiple values. Figure 2b illustrates how the fraction of k-neighbors considered for propagation varies with α. Generally, a larger value leads to stronger localization and the shape gets closer to a step function as for k-neighborhoods. Figure 5 additionally shows the average performance on CiteSeer and Cora. For single α, small values in $\{0.05, 0.1, 0.2\}$ achieve best accuracy. For larger α, runtime drops considerably but at the cost of decreased accuracy and larger variance. Among multi-scale aggregations, *sum* performs best. It slightly improves the performance over single alphas and provides additional robustness, producing smaller variance and no outlier scores. Runtime is very close to the smallest single α considered. The remaining aggregation functions lead to increased runtime and *max* does not even lead to an increase of accuracy.

4.6 Influence of Sparsity

Similarly as α, the approximation threshold ε controls the effective neighborhood size considered for propagation. We study the effect on our models with a similar setup as above. Figure 2c illustrates how a larger value of ε leads to stronger sparsification of APPR neighborhoods. Variation leads to a shift of the curve, indicating that neighbors with small visiting probabilities are discarded mostly from k-neighborhoods with moderate or large k. Figure 5 shows that accuracy remains relatively stable on CiteSeer and Cora, decreasing monotonically for increasing ε. Simultaneously, runtime decreases steadily. While smaller ε provide marginally better accuracy, our results suggest that ε may be increased safely to allow for faster runtime and to account for limited GPU memory.

5 Conclusion

We presented a novel push-based asynchronous neural message passing algorithm which allows for efficient feature aggregation over adaptive node neighborhoods. A multi-scale approach additionally leverages correlations on increasing levels of locality and variants of our model capture different inductive bias. Semi-supervised node classification experiments on five real-world benchmark datasets exhibit consistent improvements of our models over all competitors with statistical significance, demonstrating the effectiveness of our approach. Ablation studies investigate the influence of varying locality and sparsity parameters as well as combinations of multi-scale representations. In future work, we intend to investigate additional instances of the push-based message passing framework, extensions to dynamic graphs and applications to further tasks such as link prediction and graph classification.

ACKNOWLEDGEMENTS

This work was done during an internship at CT RDA BAM IBI-US, Siemens Corporate Technology, Princeton, NJ, USA.

REFERENCES

[1] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S Mirrokni, and Shang-Hua Teng, 'Local computation of pagerank contributions', *International Workshop on Algorithms and Models for the Web-Graph*, (2007).

[2] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al., 'Relational inductive biases, deep learning, and graph networks', *arXiv preprint arXiv:1806.01261*, (2018).

[3] Pavel Berkhin, 'Bookmark-coloring algorithm for personalized pagerank computing', *Internet Mathematics*, (2006).

[4] Felix Borutta, Julian Busch, Evgeniy Faerman, Adina Klink, and Matthias Schubert, 'Structural graph representations based on multi-scale local network topologies', *WI*, (2019).

[5] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst, 'Geometric deep learning: going beyond euclidean data', *IEEE Signal Processing Magazine*, (2017).

[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun, 'Spectral networks and locally connected networks on graphs', *ICLR*, (2014).

[7] Shaosheng Cao, Wei Lu, and Qiongkai Xu, 'Grarep: Learning graph representations with global structural information', *CIKM*, (2015).

[8] Jie Chen, Tengfei Ma, and Cao Xiao, 'Fastgcn: fast learning with graph convolutional networks via importance sampling', *ICLR*, (2018).

[9] Hanjun Dai, Zornitsa Kozareva, Bo Dai, Alex Smola, and Le Song, 'Learning steady-states of iterative algorithms over graphs', *ICML*, (2018).

[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst, 'Convolutional neural networks on graphs with fast localized spectral filtering', *NeurIPS*, (2016).

[11] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams, 'Convolutional networks on graphs for learning molecular fingerprints', *NeurIPS*, (2015).

[12] Evgeniy Faerman, Felix Borutta, Julian Busch, and Matthias Schubert, 'Semi-supervised learning on graphs based on local label distributions', *arXiv preprint arXiv:1802.05563*, (2018).

[13] Matthias Fey and Jan Eric Lenssen, 'Fast graph representation learning with pytorch geometric', *arXiv preprint arXiv:1903.02428*, (2019).

[14] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl, 'Neural message passing for quantum chemistry', *ICML*, (2017).

[15] Aditya Grover and Jure Leskovec, 'node2vec: Scalable feature learning for networks', *SIGKDD*, (2016).

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec, 'Inductive representation learning on large graphs', *NeurIPS*, (2017).

[17] Glen Jeh and Jennifer Widom, 'Scaling personalized web search', *WWW*, (2003).

[18] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley, 'Molecular graph convolutions: moving beyond fingerprints', *Journal of computer-aided molecular design*, (2016).

[19] Diederik P Kingma and Jimmy Ba, 'Adam: A method for stochastic optimization', *ICLR*, (2015).

[20] Thomas N Kipf and Max Welling, 'Variational graph auto-encoders', *NeurIPS Bayesian Deep Learning Workshop*, (2016).

[21] Thomas N Kipf and Max Welling, 'Semi-supervised classification with graph convolutional networks', *ICLR*, (2017).

[22] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann, 'Predict then propagate: Graph neural networks meet personalized pagerank', *ICLR*, (2019).

[23] John Boaz Lee, Ryan Rossi, and Xiangnan Kong, 'Graph classification using structural attention', *SIGKDD*, (2018).

[24] John Boaz Lee, Ryan A Rossi, Sungchul Kim, Nesreen K Ahmed, and Eunyee Koh, 'Attention models in graphs: A survey', *TKDD*, (2019).

[25] Qimai Li, Zhichao Han, and Xiao-Ming Wu, 'Deeper insights into graph convolutional networks for semi-supervised learning', *AAAI*, (2018).

[26] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel, 'Gated graph sequence neural networks', *ICLR*, (2016).

[27] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S Zemel, 'Lanczosnet: Multi-scale deep graph convolutional networks', *ICLR*, (2019).

[28] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein, 'Geometric deep learning on graphs and manifolds using mixture model cnns', *CVPR*, (2017).

[29] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe, 'Weisfeiler and leman go neural: Higher-order graph neural networks', *AAAI*, (2019).

[30] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU, 'Query-driven active surveying for collective classification', *MLG*, (2012).

[31] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov, 'Learning convolutional neural networks for graphs', *ICML*, (2016).

[32] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd, 'The pagerank citation ranking: Bringing order to the web.', Technical report, Stanford InfoLab, (1999).

[33] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena, 'Deepwalk: Online learning of social representations', *SIGKDD*, (2014).

[34] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang, 'Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec', *WSDM*, (2018).

[35] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini, 'The graph neural network model', *IEEE Transactions on Neural Networks*, (2009).

[36] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling, 'Modeling relational data with graph convolutional networks', *ESWC*, (2018).

[37] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad, 'Collective classification in network data', *AI magazine*, (2008).

[38] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann, 'Pitfalls of graph neural network evaluation', *NeurIPS Relational Representation Learning Workshop*, (2018).

[39] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei, 'Line: Large-scale information network embedding', *WWW*, (2015).

[40] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li, 'Attention-based graph neural network for semi-supervised learning', *arXiv preprint arXiv:1803.03735*, (2018).

[41] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio, 'Graph attention networks', *ICLR*, (2018).

[42] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger, 'Simplifying graph convolutional networks', *ICML*, (2019).

[43] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka, 'How powerful are graph neural networks?', *ICLR*, (2019).

[44] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Kenichi Kawarabayashi, and Stefanie Jegelka, 'Representation learning on graphs with jumping knowledge networks', *ICML*, (2018).

[45] Muhan Zhang and Yixin Chen, 'Link prediction based on graph neural networks', *NeurIPS*, (2018).

NF-GNN: Network Flow Graph Neural Networks for Malware Detection and Classification

Julian Busch
busch@dbs.ifi.lmu.de
LMU Munich
Munich, Germany

Anton Kocheturov
anton.kocheturov@siemens.com
Siemens Technology
Princeton, NJ, USA

Volker Tresp
volker.tresp@siemens.com
Siemens AG
Munich, Germany

Thomas Seidl
seidl@dbs.ifi.lmu.de
LMU Munich
Munich, Germany

ABSTRACT

Malicious software (malware) poses an increasing threat to the security of communication systems as the number of interconnected mobile devices increases exponentially. While some existing malware detection and classification approaches successfully leverage network traffic data, they treat network flows between pairs of endpoints independently and thus fail to leverage rich communication patterns present in the complete network. Our approach first extracts flow graphs and subsequently classifies them using a novel edge feature-based graph neural network model. We present three variants of our base model, which support malware detection and classification in supervised and unsupervised settings. We evaluate our approach on flow graphs that we extract from a recently published dataset for mobile malware detection that addresses several issues with previously available datasets. Experiments on four different prediction tasks consistently demonstrate the advantages of our approach and show that our graph neural network model can boost detection performance by a significant margin.

CCS CONCEPTS

• Security and privacy → Intrusion/anomaly detection and malware mitigation; **• Computing methodologies → Supervised learning by classification**; **Neural networks**; **Anomaly detection**.

KEYWORDS

Graph Neural Networks, Malware Detection

ACM Reference Format:
Julian Busch, Anton Kocheturov, Volker Tresp, and Thomas Seidl. 2021. NF-GNN: Network Flow Graph Neural Networks for Malware Detection and Classification. In *33rd International Conference on Scientific and Statistical Database Management (SSDBM 2021), July 6–7, 2021, Tampa, FL, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3468791.3468814

1 INTRODUCTION

Malicious software (malware) poses a significant threat to the security of information technology (IT) and operational technology (OT) in private and corporate environments. Along with an increasing degree of digitalization and the rise of new technologies such as the Internet of Things (IoT), an increasing number of devices, including mobile devices, such as smartphones, and Industrial Control Systems (ICS), become connected and thus are potential targets for attacks. Accurate detection and classification of malware is thus a vital task to ensure the security of such systems.

In this work, we focus on dynamic malware detection and classification. In contrast to static methods, which analyze a candidate application's source code or the structure of its executable, dynamic methods execute the application in a controlled environment and analyze dynamic behavior that can usually not be extrapolated from static data. More specifically, we consider network traffic data generated during the execution of the application, which provides valuable insights into the dynamic and potentially malicious behavior affecting that application. However, in contrast to existing works, which classify individual network flows between two endpoints, i.e., aggregated communication between the two endpoints during some time frame, we construct a communication graph from all recorded network flows between any two endpoints during that time frame to obtain a rich representation of communication in the network. To the best of our knowledge, no existing work has considered this setting so far.

While classical machine learning methods have proven successful for malware detection and classification [15], deep learning approaches have not been studied as extensively. Deep learning methods offer an additional advantage of automatically learning suitable feature representations of the input data optimized for the task at hand, in contrast to traditional feature engineering approaches. Our novel edge feature-based graph neural network model learns suitable representations from extracted network flow graphs. Along with a base model for learning graph representations, we propose three derived model variants, a graph classifier, a graph autoencoder, and a one-class graph neural network, which are able to perform supervised malware detection and classification and unsupervised malware detection, respectively.

We evaluate our approach on a graph dataset that we extract from network flow features obtained from traffic generated by android applications executed on real mobile devices. The original flow features are provided by [26]. The data was collected in a carefully designed environment to account for several common defects observed in previously used datasets. Instead of classifying individual network flows, as the original work proposes, we follow our graph-based approach and construct a flow graph for each execution of a candidate application. Experiments on four different prediction tasks, including supervised binary, category and family classification, and unsupervised detection, consistently demonstrate the significantly superior detection performance of our approach. Our neural network model additionally boosts performance compared to baseline models, even in settings with unlabeled data and small amounts of available training data.

We summarize our contributions as follows:

- We propose, to the best of our knowledge, the first graph-based approach to network traffic-based malware detection and classification.
- We propose a novel method for extracting directed edge-attributed flow graphs from sets of network flows recorded in a monitored network.
- We propose a novel graph neural network model that effectively learns representations from these graphs, utilizing the graph topology and edge attributes.
- We provide an extensive experimental evaluation on a novel flow graph dataset considering four different supervised and unsupervised detection tasks.

2 RELATED WORK

While static malware detection and classification methods analyze a candidate application's source code or the structure of its executable file, dynamic methods execute the application in a controlled environment and analyze its behavior. Such behavior is difficult and often impossible to extrapolate from static data. Further, static methods are commonly vulnerable to obfuscation techniques modifying the code or structure of the executable. Our approach is a dynamic one, focusing on network traffic data generated by a particular candidate application during execution.

A comprehensive overview of existing machine learning methods for static and dynamic malware detection and classification is provided by a recent survey [15]. Methods relying on network traffic data mainly differ by which specific features are extracted and which machine learning algorithm is used. To the best of our knowledge, all existing approaches focus on classifying individual network traffic between two endpoints in a network, possibly at different resolutions. For instance, [5] describes different resolution levels, ranging from single transactions over sessions to flows and conversations. Thereby, a flow describes traffic associated with a specific (`Source-IP, Source-Port, Destination-IP, Destination-Port, Protocol`)-tuple within a particular time frame and a conversation aggregates flows with the same Source- and Destination-IP endpoints. Instead of focusing on bilateral flows or conversations between two endpoints, our approach extracts a graph from flows between any two endpoints in a monitored network during a specific time frame to obtain a richer representation of communication in the network.

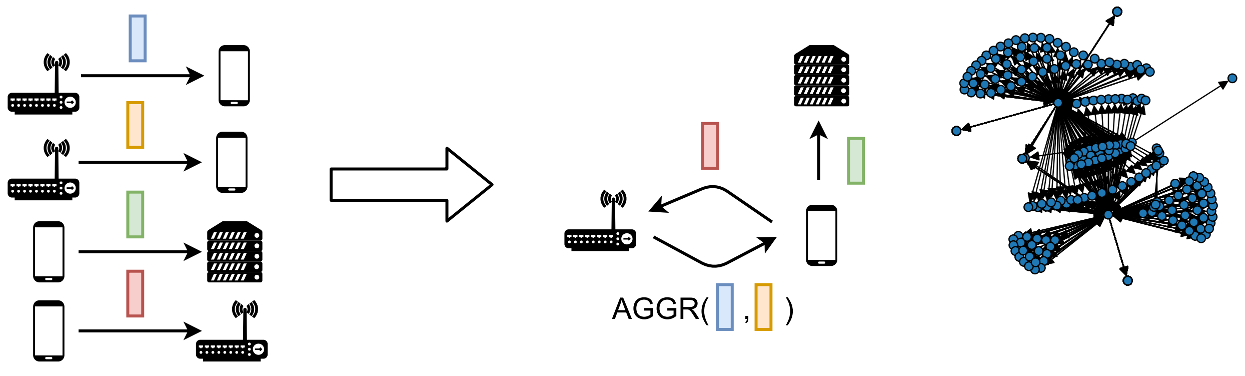
Existing methods typically report high performance on datasets that exhibit various common defects. The recently published *CICAndMal2017* dataset [26] has addressed these issues by providing a sufficient number of malware samples from diverse malware categories and families with a realistic distribution of benign and malicious applications. Additionally, each application is executed on an actual physical device instead of an emulator or a virtual machine to accurately capture its actual behavior. Instead of classifying network flows individually as proposed by the authors, we instead extract network flow graphs from the given network traffic features.

The results reported in [26] have been improved by adding new dynamic API-call features and combining a dynamic prediction model with a static prediction model [34]. Our model could be combined with a static prediction model in a similar fashion. Improved accuracy could also be achieved by predicting labels for conversations instead of individual flows [1], further mitigating the effect of port randomization techniques. Our approach takes an additional step ahead by classifying flow graphs modeling the communication between all endpoints instead of only pairs of endpoints. Two further works [6, 12] reported malware detection results only for a subset of the whole dataset.

Existing graph-based approaches to malware detection and classification mostly focus on static analysis, considering function call graphs [18, 20, 23] or control graphs [4, 14]. Dynamic graph-based approaches include [2], where graphs are constructed from instruction traces collected during execution, and [22], where system call graphs are considered. The latter work is most related to our approach since a *Graph Convolutional Neural Network (GCN)* [25] is used to learn node features. To the best of our knowledge, no existing work has considered graph-based approaches based on dynamically generated network traffic data.

Deep learning methods have not been investigated as extensively as classical machine learning methods for network traffic-based approaches. Existing methods consider detection of endpoints generating malicious traffic [29], intrusion detection [28], or ransomware detection [6]. To the best of our knowledge, no existing deep learning-based malware detection and classification method considers network flow graphs.

While our proposed approach includes a novel graph neural network model for malware detection and classification, it could be more generally employed to solve other graph classification and graph anomaly detection tasks. Graph neural networks [3, 7, 8, 16, 25, 36–38, 40] have recently become a de-facto standard for machine learning on graphs. The majority of these methods can be described in a message-passing framework, as detailed in [16]. The main idea is to iteratively propagate feature vectors through the graph by passing messages between nodes such that a node's representation depends on feature vectors appearing in its neighborhood. While most existing graph neural network models consider only node attributes, our model focuses on edge attributes instead. Some existing models consider edge attributes [17, 21, 24, 31, 33, 39], but none of these models is directly applicable to our setting since they either consider multi-relational graphs or focus on node attributes and only utilize edge features to improve message passing between nodes. Other existing models [27, 32, 42] focus on more specific tasks which differ from our setting.



(a) Network Flow Graph Extraction (b) Real Network Flow Graph

Figure 1: From a set of network flows recorded during a specific time frame, a flow graph is constructed by adding directed edges between all pairs of endpoints that communicated with at least one flow. Each flow is described by a feature vector summarizing its corresponding network traffic. Edges in the flow graph are annotated with these feature vectors, where feature vectors of parallel flows are aggregated. The topology of a real flow graph extracted from network traffic generated during execution of a *FakeAV* Scareware application is shown in (1b).

3 NETWORK FLOW GRAPH EXTRACTION

To decide whether a particular candidate instance of an application is malicious, we collect all network traffic generated during the execution of that application instance within a given time interval after installation. The resulting data consists of a set of network flows described by feature vectors that can be extracted from `pcap`-files using tools such as *CICFlowMeter* [13]. Thereby, each flow F describes network traffic associated with one particular `(Source-IP, Source-Port, Destination-IP, Destination-Port, Protocol)`-tuple during the considered time frame and has a feature vector $f \in \mathbb{R}^d$ attached to it. Typical features include the number of packets sent, mean and standard deviation of the packet length, or minimum and maximum interarrival time of the packets.

From the resulting set of flows $\mathcal{F}$, we extract a flow graph, where the nodes correspond to endpoints in the network and edges model communication between these endpoints. Instead of considering `(IP, Port)`-tuples, we factor out the port information and consider IP-endpoints for two main reasons:

(1) Apart from standard ports, port selection is often arbitrary and could even be subject to port randomization techniques, leading to arbitrary and potentially misleading graph structure.
(2) Empirically, we found `(IP, Port)`-graphs to be very sparse and rather uninformative. In comparison, IP-graphs exhibit much more interesting topologies.

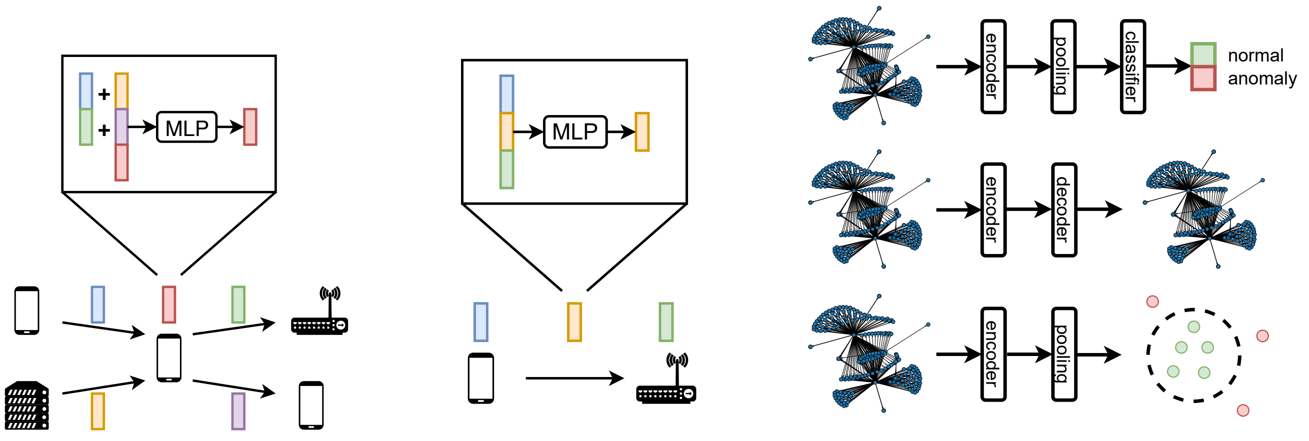
More specifically, from a set of flows $\mathcal{F}$, we extract a directed graph $G = (V, E)$ where the nodes correspond to endpoints involved in any flow $F \in \mathcal{F}$ and a directed edge is added for all pairs (s_i, t_i) for which there exists a flow $F_i \in \mathcal{F}$ with source and target IP s_i and t_i, respectively. The feature vector assigned to this edge aggregates the feature vectors $f_i \in \mathbb{R}^d$ of all flows F_i along this edge using a set of five aggregation functions. For each feature, the shape of the distribution of values along the edge is described using the first four moments, namely the mean, standard deviation, skew and kurtosis, and the median value. The aggregate values are computed for each feature and then concatenated, resulting in a feature vector $x_i \in \mathbb{R}^{5d}$ for each edge $e_i \in E$. The flow graph extraction procedure is illustrated in Figure 1a. Figure 1b shows an exemplary graph extracted from real data.

Intuitively, the resulting graph captures how network traffic flows between different endpoints in the monitored network during a specific time frame. The graph structure reveals where traffic is flowing, and the additional edge attributes describe how it is flowing. Connecting individual flows in a graph provides a much richer relational representation compared to treating flows individually. Thus, we expect models learning from these graphs to perform significantly better at detection and classification tasks than models which classify individual flows. Our experimental results confirm this intuition.

We wish to note that such graphs could potentially be used for further applications, such as intrusion detection or identifying devices generating malicious traffic.

4 NETWORK FLOW GRAPH NEURAL NETWORKS

From a machine learning perspective, we consider two different problems: Supervised graph classification and unsupervised graph anomaly detection. To solve these problems, we introduce a novel graph neural network model, which learns expressive representations from network flow graphs, along with three different variants of that base model, a supervised graph classifier, an unsupervised graph autoencoder, and an unsupervised one-class graph neural network. The different model variants are illustrated in Figure 2c.



(a) Node Update **(b) Edge Update** **(c) Model Variants**

Figure 2: Our model learns how endpoints in the network communicate with each other by sequentially updating node (2a) and edge feature vectors (2b) by neural message passing. The learned representations are used by our three model variants, a graph classifier, a graph autoencoder, and a graph one-class neural network, to solve supervised and unsupervised detection tasks (2c).

While we focus on malware detection and classification in this paper, the proposed models can be employed for general learning problems on (directed) graphs with edge attributes.

4.1 Problem Setting

Formally, we are given a set of graphs $\mathcal{G} = \{G_1, \dots, G_N\}$, a set of edge feature matrices $X_i \in \mathbb{R}^{m_i \times d}$, $i = 1, \dots, N$, where $m_i = |E_i|$, and a label matrix $\hat{Y} \in \{0,1\}^{N \times c}$ providing a class label for each graph, where c is the number of classes. For the sake of simplicity, we restrict ourselves to directed graphs in the following, though our models can be applied to undirected graphs in a straightforward fashion. Further, it is not required that the input graphs share the same topology,

In a supervised anomaly detection setting, class labels could be binary (normal vs. anomalous) or multi-class (normal class and different categories or families of anomalies). For supervised classification, a labeled training dataset is given as described above, and the task is to train a model which accurately predicts labels for new graphs not seen by the model during training. For unsupervised anomaly detection, the training set is not labeled and consists of normal data and (usually a relatively small fraction of) anomalous data. The model is required to learn a concept of normality from the training data and correctly classify new graphs as either being normal or anomalous.

4.2 Learning Representations of Network Flow Graphs

Each input instance for our model is a directed graph $G = (V, E)$ with adjacency matrix $A \in \{0,1\}^{n \times n}$ and an edge feature matrix $X \in \mathbb{R}^{m \times d}$, where $m := |E|$. The representation learning part of our model computes latent representations of the edges and nodes in the graph and finally outputs a latent feature vector $h^{(1)} \in \mathbb{R}^h$ for each node in the graph. Such a vector intuitively describes how the corresponding endpoint interacts with other endpoints in the network. Depending on the availability of labels, variants of our model compute either predictions or an anomaly score for an input graph from its latent node feature vectors. The model is trained end-to-end such that the latent representations are optimized towards the specific task. Given an input graph with edge attributes, our model performs the following feature transformation and propagation steps to sequentially compute latent representations of the graph's nodes and edges:

$$E^{(0)} = f_1(X) \qquad \in \mathbb{R}^{m \times h} \tag{1}$$

$$H^{(0)} = f_2\left(\left[\hat{\hat{\mathrm{B}}}_{in}\, E^{(0)}, \hat{\hat{\mathrm{B}}}_{out}\, E^{(0)}\right]\right) \qquad \in \mathbb{R}^{n \times h} \tag{2}$$

$$E^{(1)} = f_3\left(\left[\hat{\hat{\mathrm{B}}}_{in}^{T}\, H^{(0)}, \hat{\hat{\mathrm{B}}}_{out}^{T}\, H^{(0)}, E^{(0)}\right]\right) \qquad \in \mathbb{R}^{m \times h} \tag{3}$$

$$H^{(1)} = f_4\left(\left[\hat{\hat{\mathrm{B}}}_{in}\, E^{(1)}, \hat{\hat{\mathrm{B}}}_{out}\, E^{(1)}, H^{(0)}\right]\right) \qquad \in \mathbb{R}^{n \times h}, \tag{4}$$

where $[\cdot, \cdot]$ denotes concatenation and $f_1, \dots, f_4$ are Multi-Layer Perceptrons (MLPs) with appropriate input and output dimensions. As per default, we use single-layer MLPs

$$f_i(X) = q\left(XW_i + b_i\right), \tag{5}$$

where W_i and b_i are the learnable parameters of the model and q is a non-linear activation. We use *ReLU* activations and add batch normalization. The propagation matrices $\hat{\hat{\mathrm{B}}}_{in}, \hat{\hat{\mathrm{B}}}_{out} \in \mathbb{R}^{n \times m}$ are obtained from the node-edge incidence matrices $B_{in}, B_{out} \in \{0,1\}^{n \times m}$ with

$$(B_{in})_{ij} = \begin{cases} 1 & \text{if} \quad \exists v_k \in V : e_j = (v_k, v_i) \\ 0 & \text{else} \end{cases} \tag{6}$$

and

$$(B_{out})_{ij} = \begin{cases} 1 & \text{if} \quad \exists v_k \in V : e_j = (v_i, v_k) \\ 0 & \text{else} \end{cases}, \tag{7}$$

indicating in- and out-going edges for each node, by substituting non-zero entries with normalized edge weights. Normalization is applied to preserve the scale of the feature vectors. In particular, we apply symmetric normalization to the adjacency as in [25] before computing the node-edge incidence matrices, where the normalized adjacency matrix is given as $\hat{\tilde{A}} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ with $\tilde{A} = A + I$ and degree matrix $\tilde{D}$. Self-loops added for normalization are removed again such that the graph structure remains unchanged. Illustrations of the performed node and edge feature updates are provided in Figure 2a and 2b, respectively.

The first network layer learns how endpoints interact with each other directly by first applying a learnable feature transformation to the original edge feature vectors (Equation 1) and subsequently computing node representations by aggregating feature vectors from neighboring edges (Equation 2). Notably, incoming and outgoing traffic is modeled separately for each node.

The second layer enables the model to learn how endpoints communicate indirectly with their 2-hop neighbors. In a first step, the edge features are updated again by transforming the concatenated feature vectors of the source and destination node and the edge features from the previous layer (Equation 3). Concatenating the edge features from the previous layer as residual connections [19] gives this layer direct access to previously learned features and aids in optimization. Such skip-connections have shown to improve the performance of graph neural networks when applied to node features [41], motivating us to apply them to edge features as well. The edge feature update is followed by an update of the node features using features of incoming and outgoing edges and skip-connected node features from the first layer (Equation 4). These node representations constitute the final output of our representation learning module.

In principle, one could add more layers to the model in a similar fashion to model interaction between more distant endpoints. However, the flow graphs considered in this paper usually have a relatively small diameter, such that additional layers might not result in improved performance but rather lead to over-fitting. In our experiments, we observed the best performance with either one or two layers.

4.3 Network Flow Graph Classifier

For supervised graph classification, we append two more layers to the representation learning module. First, a pooling layer aggregates all node feature vectors in the input graph to a single vector describing the whole graph. The second layer predicts the graph label from the pooled graph representation,

$$h = \text{pool}\left(H^{(1)}\right) \qquad \in \mathbb{R}^h \tag{8}$$

$$y = \text{softmax}\left(Wh + b\right) \qquad \in \mathbb{R}^c\,. \tag{9}$$

Above, pool denotes a pooling function, such as element-wise mean or maximum, which aggregates all node representations into a single embedding vector for the whole graph. Predictions are computed by a dense prediction layer with learnable parameters $W \in \mathbb{R}^{c\times h}$ and $b \in \mathbb{R}^c$ and a softmax activation. The model parameters are then optimized w.r.t. the *cross-entropy* loss

$$\mathcal{L}_{CLF}(\mathcal{G}) = \frac{1}{N \cdot c} \sum_{\substack{G_i \in \mathcal{G} \\ j \in 1,\ldots,c}} -y_{ij} \log \hat{y}_{ij}. \tag{10}$$

We denote this model as *NF-GNN-CLF.*

4.4 Network Flow Graph Autoencoder

For unsupervised anomaly detection, autoencoder models often perform well in practice [11]. In general, an autoencoder consists of two neural network modules. While an encoder learns compact and expressive representations of the model inputs, a decoder is supposed to reconstruct the original inputs from their learned representations. If the model is trained with exclusively or mostly normal data, the reconstruction loss can be interpreted as an anomaly score, where instances incurring a larger reconstruction loss are considered more anomalous.

We propose a graph autoencoder model where our representation learning module acts as an encoder. The latent node representations $H^{(1)}$ are then used to reconstruct the original edge feature vectors of the graph using a decoder, which is a mirrored version of the encoder:

$$E^{(2)} = f_5\left(\left[\hat{\tilde{B}}_{in}^T H^{(1)}, \hat{\tilde{B}}_{out}^T H^{(1)}\right]\right) \qquad \in \mathbb{R}^{m\times h} \tag{11}$$

$$H^{(2)} = f_6\left(\left[\hat{\tilde{B}}_{in} E^{(2)}, \hat{\tilde{B}}_{out} E^{(2)}, H^{(1)}\right]\right) \qquad \in \mathbb{R}^{n\times h} \tag{12}$$

$$E^{(3)} = f_7\left(\left[\hat{\tilde{B}}_{in}^T H^{(2)}, \hat{\tilde{B}}_{out}^T H^{(2)}, E^{(2)}\right]\right) \qquad \in \mathbb{R}^{m\times h} \tag{13}$$

$$E^{(4)} = f_8\left(E^{(3)}\right) \qquad \in \mathbb{R}^{m\times h} \tag{14}$$

If the encoder uses only a single layer, the first layer of the decoder (Equation 11–12) is dropped and the node embeddings $H^{(0)}$ are used as input instead. The model parameters are optimized w.r.t. a reconstruction loss

$$\mathcal{L}_{AE}(\mathcal{G}) = \frac{1}{N} \sum_{G_i \in \mathcal{G}} \frac{1}{m_i} \left\| X_i - E_i^{(4)} \right\|_F^2, \tag{15}$$

where $|| \cdot ||_F$ denotes the Frobenius norm. A similar loss was used in [10] to reconstruct node attributes, whereas our model operates on edge-attributed graphs. We denote this variant of our model as *NF-GNN-AE.*

4.5 One-Class Network Flow Graph Neural Network

Though autoencoder models perform well in practice, they don't optimize an anomaly detection objective directly. *Deep SVDD* [30] combines *Support Vector Data Description (SVDD)* [35] with a neural network for anomaly detection in a learned latent space. The main idea is to learn a transformation into a latent space such that most instances are mapped into a hypersphere in that space, and anomalous instances will fall outside of the hypersphere.

We propose a one-class graph neural network consisting of our representation learning module and an additional pooling layer, which summarizes each input graph into a single feature vector, similarly as for the supervised graph classifier. The model parameters are optimized w.r.t. a one-class loss

$$\mathcal{L}_{OC}(\mathcal{G}) = \frac{1}{N} \sum_{G_i \in \mathcal{G}} \|h_i - \mu\|_2^2 + \frac{\lambda}{2} \sum_{i=1}^{4} \|W_i\|_F^2, \qquad (16)$$

where $\mu \in \mathbb{R}^h$ denotes the center of the hyper-sphere in latent space. The center is initialized with the mean embedding vector of all graphs in $\mathcal{G}$ after the first forward-pass and does not change thereafter. The second part of the loss regularizes the model parameters to limit model complexity. Bias vectors have been removed from all layers to prevent trivial solutions [30]. We denote this variant of our model as *NF-GNN-OC*.

5 EXPERIMENTS

We evaluate our approach on a graph dataset that we extract from the network traffic data provided by the *CICAndMal2017* dataset [26]. We focus on this datasets since it addresses several common defects shared by other existing datasets, allowing for a realistic and meaningful evaluation. These issues are addressed by providing a sufficient number of malware samples from diverse malware categories and families with a realistic distribution of benign and malicious applications. To accurately capture dynamic behavior, each application is executed on an actual physical device instead of an emulator or a virtual machine.

5.1 Dataset Preparation

Our extracted dataset consists of 2126 samples, where each sample corresponds to one instance of an android application installed and executed on a mobile phone. For each sample, all network flows within the network during execution are captured. For each flow, 80 features are recorded, including, e.g., the number of packets sent, mean and standard deviation of the packet length, and minimum and maximum interarrival time of the packets. For a more detailed description of the data collection process, we refer to [26].

For each sample, 3 different labels are available, a binary label indicating whether the application is malicious or not, a category label with 5 possible values indicating the general class of malware, and a family label with 36 different values indicating the specific type of malware. Malware families with fewer than 9 samples have been removed to ensure a reasonable split into train, validation, and test sets. Consequently, for the family prediction task, only 2071 samples are available.

For each sample, we extract a graph as described in Section 3 and remove the flow-id, timestamp, and endpoint IP and port information from the feature set. Additionally, we remove all features that are constant among all edges of all graphs, leaving 330 edge features.

To be able to compare against baselines apart from flow- and conversation-based methods and our graph neural network models, we construct additional datasets, which represent each sample by a single feature vector instead of a graph. We consider three different feature sets:

(1) **Flow Features** For each sample, we aggregate the features of all flows for this sample using the same aggregation functions we used for deriving the edge feature vectors and concatenate the aggregates to a 318-dimensional feature vector.
(2) **Graph Features** To evaluate the importance of the graph topology for the baseline models, we extract a set of structural features from each graph. In particular, we extract 2 global features (global clustering coefficient and assortativity) and 8 local node-features (degree, number of 2-hop neighbors, clustering coefficient, avg. neighbor degree, avg. neighbor clustering coefficient, number of edges in egonet, number of edges leaving egonet, betweenness centrality) that are aggregated over all nodes in the graph, again using the same aggregation functions. All extracted graph features are concatenated, leading to a 40-dimensional feature vector.
(3) **Combined Features** To provide access to both types of features, we concatenate the flow and graph features to a combined 358-dimensional feature vector for each sample.

Again, all constant feature columns have been removed. All feature matrices, including the edge feature matrices for our model, are standardized before training.

5.2 Supervised Malware Detection and Classification

We consider three supervised tasks, binary, category, and family classification. To ensure a fair and unbiased comparison, we follow a rigorous evaluation protocol.

5.2.1 Experimental Setup. First, we split the dataset into a train, validation, and test part. Each model is trained on the training set, hyper-parameters are chosen based on validation set performance using a grid search, and results are reported on the test set using the optimal hyper-parameter values. All experiments are repeated on 30 randomly generated splits, and mean and standard deviation of the results are reported.

To ensure a balanced training set, we sample 100, 25, and 5 samples per class for training for binary, category, and family classification, respectively. For binary and category classification, 5% of the remaining samples are sampled in a stratified fashion for validation. The remaining samples are used for testing. For family prediction, 20% of the non-training samples are chosen for validation to account for smaller class sizes. Stratification ensures that smaller classes are represented appropriately in the validation set.

Again, to account for class imbalance, we report weighted precision, recall, and F1 scores. In all considered settings, we checked that our models outperform the respective best competitor with statistical significance at $P < 1e - 5$ according to a Wilcoxon signed-rank test.

We compare our model against 7 baseline algorithms with different inductive bias, *Support Vector Machine (SVM)* with linear and RBF-kernel, *k-Nearest Neighbor Classifier (KNN)*, *Decision Tree (DT)*, *Random Forest (RF)*, *Adaboost (ADA)* and a *Multi-Layer Perceptron (MLP)* with up to two dense layers and *ReLU* and *softmax* activations. Additionally, we compare our graph-based approach against two existing flow-based approaches [26, 34] and one conversation-based approach [1]. Since no source code has been made available,

	Weighted Recall			Weighted Precision		
	Binary	**Category**	**Family**	**Binary**	**Category**	**Family**
Flows [26]	88.30	48.50	25.50	85.80	49.90	27.50
Flows + Static + API Calls [34]	95.30	81.00	61.20	95.30	83.30	59.70
Conversations [1]	89.00	79.64	66.59	86.65	80.20	67.21
SVM-LIN	96.26 ± 2.12	72.83 ± 10.03	28.74 ± 15.56	96.72 ± 1.49	87.31 ± 0.80	90.35 ± 0.37
SVM-RBF	96.20 ± 1.51	85.42 ± 3.84	49.96 ± 16.20	96.64 ± 1.06	89.13 ± 2.18	91.52 ± 0.98
KNN	95.71 ± 2.05	78.10 ± 12.29	42.53 ± 17.86	95.75 ± 1.96	87.08 ± 2.40	90.92 ± 0.76
DT	92.65 ± 4.13	73.42 ± 12.19	45.98 ± 30.81	93.79 ± 2.80	85.63 ± 6.59	85.72 ± 18.57
RF	95.85 ± 2.00	84.30 ± 8.24	56.06 ± 19.60	96.24 ± 1.60	90.32 ± 2.60	91.67 ± 0.99
ADA	96.38 ± 1.62	76.02 ± 12.67	42.17 ± 28.82	96.67 ± 1.31	83.97 ± 6.72	87.88 ± 13.35
MLP	97.19 ± 1.19	85.69 ± 4.46	49.56 ± 17.75	97.29 ± 1.07	89.28 ± 2.49	90.23 ± 4.32
NF-GNN-CLF	**99.42 ± 0.45**	**95.41 ± 1.48**	**91.37 ± 8.39**	**99.44 ± 0.44**	**96.14 ± 1.07**	**93.62 ± 2.52**

Table 1: Weighted recall and precision scores for the three supervised tasks. For competing methods, we report the scores provided by the respective authors. For our baselines and our model, scores are reported in terms of mean and standard deviation over 30 independent data splits.

we report the scores provided by the respective authors. Results have been provided only for a single data split.

For all neural network models, we use early stopping based on validation set performance using a patience of 20 epochs and a maximum of 1000 epochs. The default learning rate for all MLP-models is fixed as $1e-3$. For our model, we additionally apply dropout regularization before each dense layer and on the propagation matrices. All considered hyper-parameter values for all models are provided in Appendix A.

Since the original feature vectors are rather high-dimensional, we introduce an additional hyper-parameter for each baseline model indicating whether or not to perform *Principle Component Analysis (PCA)* as feature reduction before training where 95% of the variance in the data is retained.

5.2.2 Detection and Classification Performance. Detection and classification results for all three tasks are reported in terms of weighted recall and precision in Table 1. For each of our baseline model, the best feature set (Flow, Graph, or Combined) has been chosen based on validation set performance.

While some baselines perform notably weaker than others, we can observe that all baseline models still exceed the best results reported for flow classification in [26] in terms of both recall and precision, in most cases by a large margin.

Flow classification with an added static detection model and additional API-call features [34] performs better than some of our weaker baselines, especially in terms of recall. However, this approach is outperformed in terms of precision by all our baselines on category and family prediction. Our stronger baselines can outperform this approach, sometimes even by a large margin, except in terms of recall for family classification. It is important to note that [34] use additional data compared to our baselines and our model. Our approach could, in principle, be extended to also use this additional data and thus further boost performance.

Conversation-level detection [1] performs worse than [34], except for family classification. This competitor outperforms all of our baselines in terms of recall on the family classification task. In terms of precision, however, all of our baselines outperform all competitors by a significant margin.

The competitive and sometimes even vastly superior performance of our baselines already supports the main motivation of our approach to detect and classify malware using network flow graphs. Notably, this performance has been achieved even under a rigorous evaluation protocol and using relatively small training sets. In comparison, the competitors have used training set sizes between 60% and 80%.

Our proposed graph neural network model NF-GNN-CLF can further boost performance significantly compared to the baselines. It is able to significantly exceed the performance of all competitors on all three tasks in terms of both recall and precision. In particular, compared to the best competitor, recall can be improved by 4.12%, 14.41%, and 24.78% for binary, category, and family classification,

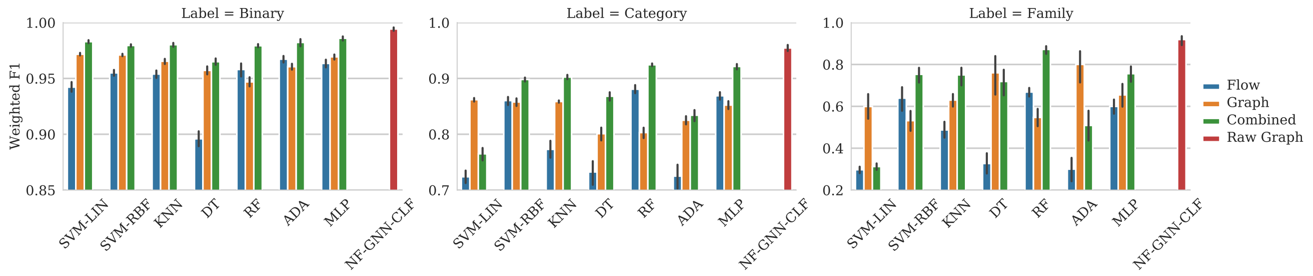


Figure 3: Weighted F1 scores for the three supervised tasks using different feature sets. Results are reported in terms of mean and standard deviation over 30 independent data splits.

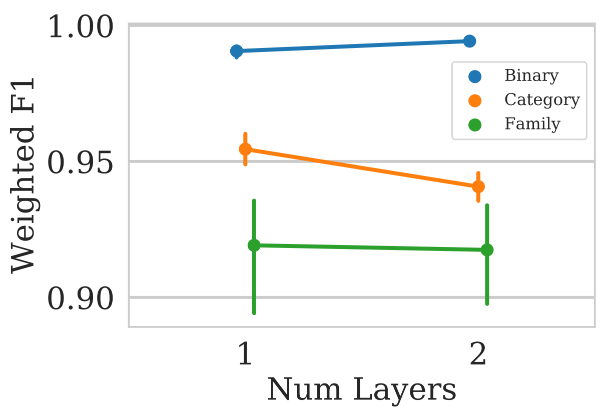


Figure 4: Performance of our model on supervised tasks with different numbers of layers.

respectively. Precision is improved by 4.14%, 12.84%, and 26.41%. We wish to note that in most cases, the best competitor [34] uses additional data that is not available to our baselines and model. Compared to the other two competitors, the performance gain is even more significant. The largest improvement is achieved for the family classification task with a performance increase of over 65% compared to flow classification [26].

5.2.3 Importance of Different Feature Sets. To get more insight into the importance of different feature sets for our baseline models, we compare their performance in terms of weighted F1 score on different feature sets in Figure 3. We can observe that almost all baseline models perform best on the combined flow and graph feature set.

As a notable exception, the linear SVM performs best on category and family classification using only the graph features, and adding flow features significantly hurts performance. However, even using only graph features, this baseline still performs worst among all baselines. Similarly, DT and ADA perform best using only graph features for family classification but are also outperformed by other baselines using the combined feature set. In most cases, the combination boosts performance significantly over the best individual feature set.

Our model operating on the raw graphs still outperforms all baselines.

5.2.4 Influence of the Number of Network Layers. To further examine the influence of the number of layers on our model's performance, we compare different choices for the supervised classification tasks in Figure 4. We can observe that modeling 2-hop interactions between endpoints can boost performance on the binary prediction tasks, while direct interactions are more crucial for the remaining two tasks. In general, performance remains relatively stable for different numbers of layers.

5.3 Unsupervised Malware Detection

We further evaluate our approach in a more realistic unsupervised setting, where no labels are available for training.

5.3.1 Experimental Setup. Our experimental setup is the same as in the supervised case with a few adjustments. To ensure a realistic distribution of benign and malicious samples, we perform stratified sampling to first split 20% of the samples for training and then 10% of the remaining samples for validation. The remaining samples are used for testing. All algorithms obtain access to the labeled validation set for hyper-parameter optimization, but training is still unsupervised. We evaluate detection performance using AUROC since it inherently adjusts for class imbalance [9].

We evaluate against a set of popular baseline algorithms for anomaly detection, *One-class SVM (OC-SVM)* with linear and RBF-kernel, *Local Outlier Factor (LOF)*, *Kernel Density Estimation (KDE)*, *Isolation Forest (IF)*, *Autoencoder with dense layers (MLP-AE)* and *One-Class Neural Network with dense layers (MLP-OC)*. Again, all considered hyper-parameter values are provided in the supplement. Since the flow- and conversation-based competitors [1, 26, 34] only consider supervised detection and classification, we are not able to include them for comparison.

5.3.2 Detection Performance. Table 2 shows the detection results for different feature sets. Results for different feature sets are additionally visualized in Figure 6. We can observe that several models report better prediction performance using only structural graph features. Thus, compared to the supervised setting, it is even more important to consider the topology of the network flow graph. While both neural network baselines, as well as KDE and IF already exhibit high detection performance, our models can again boost performance by a significant margin, demonstrating the importance

	OC-SVM-LIN	OC-SVM-RBF	LOF	KDE	IF	MLP-AE	MLP-OC	NF-GNN-AE	NF-GNN-OC
Flow	57.17 ± 4.81	70.01 ± 1.90	73.15 ± 1.22	72.50 ± 1.03	70.39 ± 1.12	71.64 ± 1.20	81.29 ± 4.07	|	|
Graph	54.60 ± 19.32	67.19 ± 2.89	58.57 ± 5.12	91.79 ± 0.66	90.73 ± 1.31	89.56 ± 0.77	94.00 ± 1.32	95.34 ± 0.85	**96.75 ± 1.22**
Combined	58.21 ± 4.83	76.07 ± 1.78	77.94 ± 1.56	86.60 ± 1.07	85.73 ± 2.21	83.04 ± 0.78	94.03 ± 4.47	|	|

Table 2: AUROC scores for unsupervised malware detection in terms of mean and standard deviation over 30 independent data splits. Our models use the raw graphs as input instead of the features extracted for the baseline models.

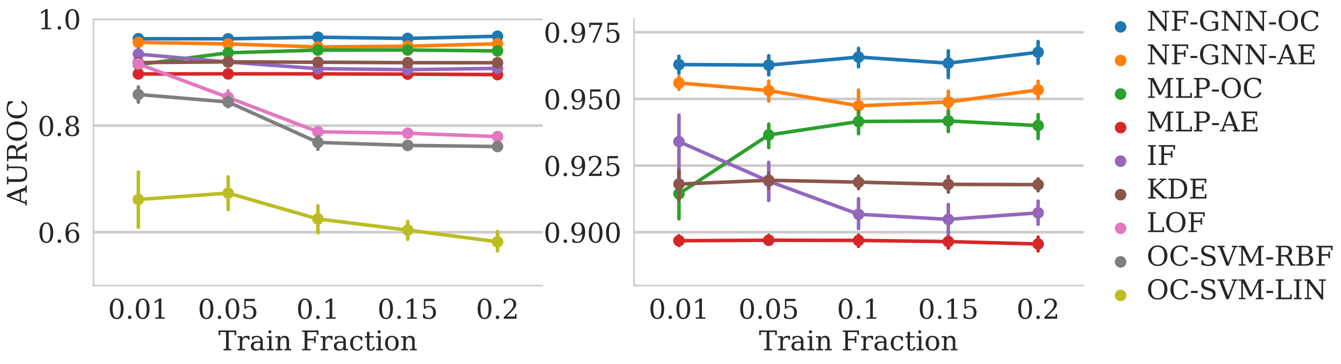


Figure 5: AUROC scores for unsupervised malware detection using different fractions of training samples. The right-hand figure provides an enlarged view of the upper part of the left-hand figure.

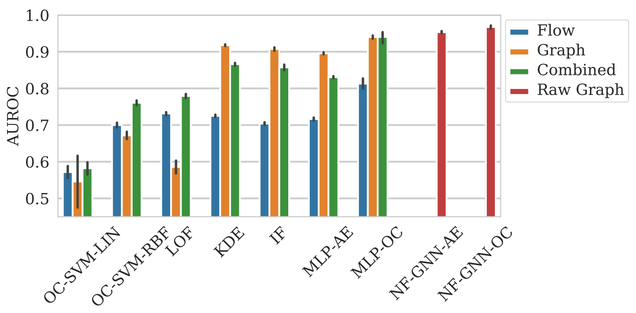


Figure 6: AUROC scores for unsupervised malware detection using different feature sets.

of learning suitable representations from network flow graphs. We wish to emphasize that the performance of our unsupervised models, NF-GNN-AE and NF-GNN-OC, almost matches that of their supervised binary classification counterpart, NF-GNN-CLF, even without any labels provided for training.

5.3.3 Influence of the Training Set Size. In practice, there is often a shortage of available training data, even unlabeled data. Thus, we further investigate performance using different amounts of training data where we gradually reduce the fraction of training samples from 20% to 1%. For each training set size, a new grid search is performed to determine the best hyper-parameters for each model. Figure 5 shows that while some models perform worse with more available training data, possibly due to overfitting to anomalies in the training set, especially MLP-OC and NF-GNN-OC benefit from more training data. For all training set sizes, both of our models consistently outperform all competing baselines.

6 CONCLUSION

We proposed a novel network flow graph-based approach to malware detection and classification, where we monitor network traffic generated by a candidate application and extract a flow graph, which models communication between devices during the considered time frame. In addition, we proposed a novel edge feature-based graph neural network model along with three different model variants for supervised and unsupervised settings. Empirically, we found that even baseline models operating on manually extracted graph and flow features perform very well in all settings. In addition, our proposed models automatically learn suitable representations of network flow graphs and can boost performance even further, significantly outperforming all competitors on all tasks. Ablation studies examined the influence of different feature sets, the number of network layers, and training set size on detection and classification performance. In future work, we plan to consider additional network architectures, such as attention, model temporal dynamics, and consider explainability of model decisions.

ACKNOWLEDGMENTS

This work was done during an internship at CT RDA BAM IBI-US, Siemens Technology, Princeton, NJ, USA.

REFERENCES

[1] Mohammad Abuthawabeh and Khaled Mahmoud. 2020. Enhanced android malware detection and family classification using conversation-level network traffic features. *International Arab Journal of Information Technology* 17 (2020), 607–614.

[2] Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. 2011. Graph-based malware detection using dynamic analysis. *Journal in computer Virology* 7, 4 (2011), 247–258.

[3] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).

[4] Zahra Bazrafshan, Hashem Hashemi, Seyed Mehdi Hazrati Fard, and Ali Hamzeh. 2013. A survey on heuristic malware detection techniques. In *The 5th Conference on Information and Knowledge Technology*. IEEE, 113–120.

[5] Dmitri Bekerman, Bracha Shapira, Lior Rokach, and Ariel Bar. 2015. Unknown malware detection using network traffic classification. In *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 134–142.

[6] Iram Bibi, Adnan Akhunzada, Jahanzaib Malik, Ghufran Ahmed, and Mohsin Raza. 2019. An effective android ransomware detection through multi-factor feature filtration and recurrent neural network. In *2019 UK/China Emerging Technologies (UCET)*. IEEE, 1–4.

[7] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.

[8] Julian Busch, Jiaxing Pi, and Thomas Seidl. 2020. PushNet: Efficient and Adaptive Neural Message Passing. In *24th European Conference on Artificial Intelligence (ECAI)*.

[9] Guilherme O Campos, Arthur Zimek, Jörg Sander, Ricardo JGB Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E Houle. 2016. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data mining and knowledge discovery* 30, 4 (2016), 891–927.

[10] Keting Cen, Huawei Shen, Jinhua Gao, Qi Cao, Bingbing Xu, and Xueqi Cheng. 2020. ANAE: Learning Node Context Representation for Attributed Network Embedding. *arXiv preprint arXiv:1906.08745* (2020).

[11] Raghavendra Chalapathy and Sanjay Chawla. 2019. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407* (2019).

[12] Rong Chen, Yangyang Li, and Weiwei Fang. 2019. Android malware identification based on traffic analysis. In *International Conference on Artificial Intelligence and Security*. Springer, 293–303.

[13] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. 2016. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*. 407–414.

[14] Parvez Faruki, Vijay Laxmi, Manoj Singh Gaur, and P Vinod. 2012. Mining control flow graph as api call-grams to detect portable executable malware. In *Proceedings of the Fifth International Conference on Security of Information and Networks*. 130–137.

[15] Daniel Gibert, Carles Mateu, and Jordi Planes. 2020. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications* 153 (2020), 102526.

[16] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*. PMLR, 1263–1272.

[17] Liyu Gong and Qiang Cheng. 2019. Exploiting edge features for graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9211–9219.

[18] Mehadi Hassen and Philip K Chan. 2017. Scalable function call graph-based malware classification. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. 239–248.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[20] Haodi Jiang, Turki Turki, and Jason TL Wang. 2018. DLGraph: Malware detection using deep learning and graph embedding. In *2018 17th IEEE international conference on machine learning and applications (ICMLA)*. IEEE, 1029–1033.

[21] Xiaodong Jiang, Ronghang Zhu, Sheng Li, and Pengsheng Ji. 2020. Co-embedding of Nodes and Edges with Graph Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[22] Teenu S John, Tony Thomas, and Sabu Emmanuel. 2020. Graph Convolutional Networks for Android Malware Detection with System Call Graphs. In *2020 Third ISEA Conference on Security and Privacy (ISEA-ISAP)*. IEEE, 162–170.

[23] Joris Kinable and Orestis Kostakis. 2011. Malware classification based on call graph clustering. *Journal in computer virology* 7, 4 (2011), 233–245.

[24] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. 2018. Neural relational inference for interacting systems. In *International Conference on Machine Learning*. PMLR, 2688–2697.

[25] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations (ICLR)*.

[26] Arash Habibi Lashkari, Andi Fitriah A Kadir, Laya Taheri, and Ali A Ghorbani. 2018. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *2018 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 1–7.

[27] Lu Lin and Hongning Wang. 2020. Graph Attention Networks over Edge Content-Based Channels. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1819–1827.

[28] AL-Hawawreh Muna, Nour Moustafa, and Elena Sitnikova. 2018. Identification of malicious activities in industrial internet of things based on deep learning models. *Journal of information security and applications* 41 (2018), 1–11.

[29] Paul Prasse, Lukáš Machlica, Tomáš Pevný, Jiří Havelka, and Tobias Scheffer. 2017. Malware detection by analysing network traffic with neural networks. In *2017 IEEE Security and Privacy Workshops (SPW)*. IEEE, 205–210.

[30] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. 2018. Deep one-class classification. In *International conference on machine learning*. PMLR, 4393–4402.

[31] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*. Springer, 593–607.

[32] Chao Shang, Qinqing Liu, Ko-Shin Chen, Jiangwen Sun, Jin Lu, Jinfeng Yi, and Jinbo Bi. 2018. Edge attention-based multi-relational graph convolutional networks. *arXiv preprint arXiv:1802.04944* (2018).

[33] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3693–3702.

[34] Laya Taheri, Andi Fitriah Abdul Kadir, and Arash Habibi Lashkari. 2019. Extensible android malware detection and family classification using network-flows and API-calls. In *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 1–8.

[35] David MJ Tax and Robert PW Duin. 2004. Support vector data description. *Machine learning* 54, 1 (2004), 45–66.

[36] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *6th International Conference on Learning Representations (ICLR)*.

[37] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871.

[38] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* (2020).

[39] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* (2020).

[40] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks?. In *7th International Conference on Learning Representations (ICLR)*.

[41] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*. PMLR, 5453–5462.

[42] Xikun Zhang, Chang Xu, Xinmei Tian, and Dacheng Tao. 2019. Graph edge convolutional neural networks for skeleton-based action recognition. *IEEE transactions on neural networks and learning systems* 31, 8 (2019), 3047–3060.

A HYPER-PARAMETER OPTIMIZATION

For the sake of a fair comparison, hyper-parameters of all models are optimized by a grid search based on performance on a separate validation set. For each algorithm, we consider a set of possible values for each of its adjustable hyper-parameters. While some common choices can be found in the literature, for the remaining hyper-parameters, we determine potential values, which seem most promising within the available computational budget. The considered hyper-parameter values for all supervised models can be found in Table 3. The values considered for the unsupervised models are provided in Table 4.

Algorithm	**Parameter**	**Values**
Support Vector Machine (SVM)	C	$[2^{-7}, 2^{-6}, \ldots, 2^{7}]$
	γ (RBF-kernel)	$[2^{-7}, 2^{-6}, \ldots, 2^{7}]$
k-Nearest Neighbor Classifier (KNN)	num. neighbors	$[1, 2, 3, 5, 8, 13, 21]$
Decision Tree (DT)	max. depth	$[2, 5, 10, None]$
	max. features	$[sqrt, None]$
Random Forest (RF)	num. estimators	$[10, 100, 1000]$
	criterion	$[entropy, gini]$
	max. features	$[sqrt, None]$
Adaboost (ADA)	num. estimators	$[10, 100, 1000]$
	learning rate	$[1e-3, 1e-2, 1e-1, 1]$
Multi-Layer Perceptron (MLP)	num. layers	$[1, 2]$
	num. hidden	$[16, 32, 64, 128]$
	L2-reg.	$[0, 1e-1, 1e-2, 1e-3, 1e-4]$
NF-GNN-CLF (ours)	num. layers	$[1, 2]$
	num. hidden	$[16, 32, 64, 128]$
	learning rate	$[1e-3, 1e-2]$
	dropout prob.	$[0, 0.2, 0.4, 0.6]$
	pool	$[mean, add, max]$

Table 3: Hyper-parameter values used in grid search for supervised algorithms.

Algorithm	**Parameter**	**Values**
One-class SVM (OC-SVM)	ν	$[1e-2, 1e-1]$
	γ (RBF-kernel)	$[2^{-10}, 2^{-9}, \dots, 2^{10}]$
Local Outlier Factor (LOF)	num. neighbors	$[1, 2, 3, 5, 8, 13, 21]$
Kernel Density Estimation (KDE)	bandwidth	$[2^{0.5}, 2, \dots, 2^5]$
Isolation Forest (IF)	num. estimators	$[10, 100, 1000]$
	max. features	$[256, None]$
Autoencoder (MLP-AE)	num. layers	$[1, 2]$
	num. hidden	$[16, 32, 64, 128]$
	L2-reg.	$[0, 1e-1, 1e-2, 1e-3, 1e-4]$
One-class MLP (MLP-OC)	num. layers	$[1, 2]$
	num. hidden	$[16, 32, 64, 128]$
	L2-reg.	$[0, 1e-1, 1e-2, 1e-3, 1e-4]$
NF-GNN-AE (ours)	num. layers	$[1, 2]$
	num. hidden	$[16, 32, 64, 128]$
	learning rate	$[1e-3, 1e-2]$
	dropout prob.	$[0, 0.2, 0.4, 0.6]$
	pool	$[mean, add, max]$
NF-GNN-OC (ours)	num. layers	$[1, 2]$
	num. hidden	$[16, 32, 64, 128]$
	learning rate	$[1e-3, 1e-2]$
	dropout prob.	$[0, 0.2, 0.4, 0.6]$
	pool	$[mean, add, max]$

Table 4: Hyper-parameter values used in grid search for unsupervised algorithms.

Semi-Supervised Learning on Graphs Based on Local Label Distributions

Evgeniy Faerman, Felix Borutta, Julian Busch, Matthias Schubert
Ludwig-Maximilians-Universität München
Munich, Germany
{faerman,borutta,busch,schubert}@dbs.ifi.lmu.de

ABSTRACT

Most approaches that tackle the problem of node classification consider nodes to be similar, if they have shared neighbors or are close to each other in the graph. Recent methods for attributed graphs additionally take attributes of neighboring nodes into account. We argue that the class labels of the neighbors bear important information and considering them helps to improve classification quality. Two nodes which are similar based on class labels in their neighborhood do not need to be close-by in the graph and may even belong to different connected components. In this work, we propose a novel approach for the semi-supervised node classification. Precisely, we propose a new node embedding which is based on the class labels in the local neighborhood of a node. We show that this is a different setting from attribute-based embeddings and thus, we propose a new method to learn label-based node embeddings which can mirror a variety of relations between the class labels of neighboring nodes. Our experimental evaluation demonstrates that our new methods can significantly improve the prediction quality on real world data sets.

KEYWORDS

Feature learning, Graph representations, Node embeddings, Node classification

ACM Reference Format:
Evgeniy Faerman, Felix Borutta, Julian Busch, Matthias Schubert. 2018. Semi-Supervised Learning on Graphs Based on Local Label Distributions. In *ACM Workshop@SIGKDD (MLG'18).* ACM, London, UK, 8 pages.

1 INTRODUCTION

Graphs are the most general way to represent structured data. In general, a set of entities with some given pairwise relationships between them can be modeled as a graph $G = (V, E)$ with a corresponding node set V and an edge set $E \subseteq V \times V$. Real-world examples of graph-structured data are abundant and include social networks, co-citation networks or biological networks.

In addition to the graph structure, further attribute information may be provided for the entities described by the graph nodes. In an attributed graph, each node $v_i \in V$ is associated with an attribute vector $f_i \in \mathbb{R}^d$. For instance, social network users might be enriched with personal information or documents in a co-citation network might be described by bag-of-words vectors. The increasing relevance of graph-structured data has been accompanied by an increased interest in learning algorithms which can leverage underlying graph structure to make accurate predictions for the modeled entities.

An important semi-supervised learning task on graphs is node classification, where each node $v_i \in V$ can be associated with a set of class labels (simply referred to as labels in the following) represented by a label vector $y_i \in \{0, 1\}^l$ where l is the number of possible labels. Given a set of already labeled nodes in a graph, the goal is to predict new likely labels for unlabeled nodes. The task is semi-supervised in the sense that connectivity information about the whole graph is available and at least some of the class labels are already known. In the case of attributed graphs, attributes of all nodes can additionally be used for prediction, including those of the unlabeled nodes in the graph. Important applications include recommendation in social networks, where the node labels represent user interests, or document classification in co-citation networks, where the node labels indicate associated fields of research.

Approaches for node classification on graphs may employ additional node attributes or operate on the graph structure alone. We will refer to these approaches as *attribute-based* and *connectivity-based* approaches, respectively. Among the most successful connectivity-based methods are node embedding techniques [7, 9, 13, 16, 17, 21, 32, 33, 37, 40]. An underlying assumption of these techniques is that nodes which are closely connected in the graph, should have similar labels, which is commonly referred to as homophily [25]. Our method does not rely on the homophily assumption, but is still able to relate close-by nodes. Furthermore, unlike most node embedding techniques our new approach can be used to classify nodes unseen during training. In the attribute-based setting the graph structure can be incorporated in different ways, for instance by using regularization [5, 41, 44, 45], combining attributes with node embeddings [42], or aggregating them over local neighborhoods [4, 8, 12, 20, 23, 26, 27, 39]. While regularization-based methods rely on the homophily assumption and most of them are not able to classify instances unseen during training, all other methods focus on node attributes. In addition to connectivity and node attributes, the labels available during training further provide valuable information that is in general complementary to connectivity and attribute features, and are useful to improve classification. In general learning tasks on independent and identically distributed (iid) data, labels indicate that an observation is sampled from a particular distribution. However, in a graph we have non-iid data and thus, the labels of connected objects allow for a novel use of label information which has not been exploited before for learning graph embeddings.

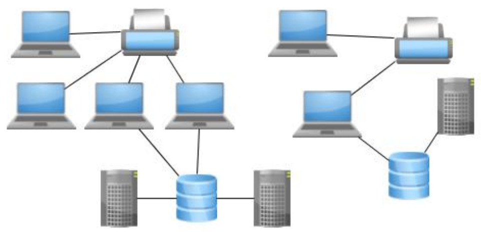


Figure 1: Consider a communication network with nodes labeled according to their device type (user, server, database, printer). Assume the labels for the database and printer nodes in the right connected component are unknown while the remaining labels are provided. Further node attributes are not given. We can observe that the roles of printers and databases are clearly defined by the labels of their neighboring nodes, e.g., printers are not connected to server nodes. Homophily-based methods would fail to classify these nodes correctly, since their labels differ from their neighbors. Further, connectivity alone does not explain the roles, since for instance the printer and database in the left part of the graph have the same degree and even their neighbors have the same degrees.

In this paper, we propose a *label-based* approach to learn a node embedding which allows for more accurate node classifications. The main idea of our approach is that there often exists a correlation between the labels of a node and the distribution of labels in its local neighborhood. Thus, considering the local label distribution when computing a node embedding can exploit this correlation to improve the descriptiveness of the learned embedding. In Figure 1, we illustrate this for a typical case for which the label of a node is determined by the labels of neighboring nodes and not by node attributes or connectivity. As an additional example, the function of a protein can be expected to correlate strongly with the functions of interacting proteins. As mentioned above, we assume that the labels of at least some of the neighboring nodes are known for each new node with unknown labels. In the majority of applications, this is realistic because new nodes usually connect to already known parts of the network. For instance, new papers usually cite established articles and new members of a social network will usually already know multiple friends in the network to connect to.

Though labels can be considered as another type of node attributes, there exists an important difference between labels and attributes which prevents attribute-based embeddings to generalize well on label information. Though the attribute values of the predicted node are allowed to be used for learning the embedding, using the node labels even in an transitive way leads to overfitting and a bad generalization performance of the learned embedding. We will discuss these issues in more detail in Section 3 and introduce a simple baseline method. In our new method, we aggregate labels from relevant nodes directly and thus, we can completely exclude any influence of the nodes' own labels. In a first step, we determine the relevant neighbors of a given node based on *Approximate Personalized PageRank (APPR)*. Since this might be an expensive task for large graphs, we use an adaption of the highly efficient algorithm from [36]. After determining the neighborhood, we compute the label distribution within the neighborhood and classify the node based on this novel representation. We compare our new representation to state-of-the-art graph embeddings based on several benchmark datasets for node classification.

The remainder of the paper is structured as follows: After providing a formal problem definition for our approach in Section 2, we introduce our new method in Section 3, starting with a discussion on the possibility of incorporating label-based features into existing models in Section 3.1. After a discussion of related work in Section 4, the performance of our model is evaluated experimentally and compared to state-of-the-art methods in Section 5. Finally, Section 6 concludes the paper and proposes directions for future work.

2 PROBLEM SETTING

We consider (possibly directed) graphs $G = (V, E)$, with node set $V = \{v_1, \ldots, v_n\}$ and edge set $E \subseteq V \times V$. A graph can be represented by an $n \times n$ adjacency matrix $\mathbf{A} = (a_{ij})_{v_i, v_j \in V}$, where $a_{ij} \in \mathbb{R}$ denotes the weight of the edge (v_i, v_j). In case of an unweighted graph, $a_{i,j} = 1$ indicates the existence and $a_{i,j} = 0$ the absence of an edge between v_i and v_j. Furthermore, we do not allow self-links, i.e., $a_{i,i} = 0$ for all nodes $v_i \in V$. In an attributed graph, additional node attributes are provided in the form of an attribute vector $f_i \in \mathbb{R}^d$ for each node v_i. The attribute information for the whole graph can be represented by an $n \times d$ attribute matrix $\mathbf{F}$, where the ith row of $\mathbf{F}$ corresponds to v_i's attribute vector f_i. Let us note that an important difference between attributes and labels is that attributes are usually known for all nodes, in particular those nodes without known labels.

Our problem setting is *semi-supervised node classification*, where the node set V is partitioned into a set of labeled nodes L and unlabeled nodes U, such that $V = L \cup U$ and $L \cap U = \emptyset$. Thereby, each node $v_i \in V$ is associated with a label vector $y_i \in \{0, 1\}^l$, where l is the number of possible labels and an entry one indicates the presence of the corresponding label for a certain node. The labels available for training can be represented by an $n \times l$ label matrix $\mathbf{Y}_{train}$, where the i'ths row of $\mathbf{Y}_{train}$ corresponds to the label vector y_i of v_i if $v_i \in L$. For unlabeled nodes, we assign constant zero vectors. The task is now to train a classifier using $\mathbf{A}$, $\mathbf{Y}_{train}$ and possibly $\mathbf{F}$ which accurately predicts y_i for each $v_i \in U$. In *multi-class* classification, each node is assigned to exactly one class, such that $y_i = e_j$ is the j's unit vector, if v_i is assigned to class j. *Multi-label* classification denotes the general case, in which each node may be assigned to one or more classes and the goal is to predict all labels assigned to a particular node.

3 SEMI-SUPERVISED LEARNING ON GRAPHS BASED ON LOCAL LABEL DISTRIBUTION

3.1 Labels as Attributes

The main idea of our approach is to learn a more descriptive node representation by incorporating the known labels in the neighborhood of a node. In the following, we will show why existing methods are not suitable to consider this information. Methods relying on neighborhood similarity [7, 9, 13, 16, 32, 37, 40] learn representations in an unsupervised manner and thus, only rely on the topology of the graph and not on attributes or labels. The *Planetoid-T* model [42] considers labels by partly enforcing the similarity between members of the same class and therefore, nodes are related to each other based only on their own labels.

Graph Neural Networks [24, 34] or Graph Convolution Networks (GCN) [4, 8, 12, 20, 23, 26, 27, 39] are special cases of a Message Passing Neural Network (MPNN) [14] which is a framework describing a family of neural network based models for attributed graphs. All MPNN methods have in common that they use some differentiable function to iteratively compute messages for each node which are passed to all its neighbors. These messages build an input to a differentiable update function which computes new node representations h:

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t),$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}).$$

Here t denotes the current iteration, h_v^t is the representation of node v in iteration t and vector h_v^0 corresponds to the input features of node v. $N(v)$ denotes the set of direct neighbors of node v, M_t is the message and U_t the set of update functions. The obvious way to integrate the neighborhood label information into an MPNN-based prediction model is to include the label information into the messages directed to the neighbors in the first iteration. However, even after removing self-links each node would receive information about its own labels already in the second iteration during training. Thus, models learned on such representations overfit on the nodes' own labels and do not generalize well in the inference step where the node labels are unknown. The same applies to directed graphs with cycles. Therefore, applying MPNN models to communicate neighboring labels is restricted to one iteration only. We use a corresponding model as a baseline for our experiments.

Note that this problem does not apply to label diffusion algorithms [15, 19, 44, 45]. However, these methods infer node labels based on majority vote in local neighborhoods and do not make use of recurrent patterns in graph.

3.2 General Approach

To present our method for semi-supervised learning on graphs using local label distributions we first outline an efficient algorithm for computing node neighborhoods based on *Approximated Personalized PageRank* (APPR). Afterwards, we describe how to create node representations based on the label distribution in the local neighborhood based on APPR. Finally, the node representations can be used as feature descriptors in arbitrary classification models.

The *Personalized PageRank* (PPR) corresponds to the *PageRank* algorithm [31], where the probabilities in the starting vector s are biased towards some set of nodes. The result is the "importance" of all nodes in the graph from the viewpoint of the nodes in s.

The *push* algorithm described in [18] and [6] is an efficient way to compute an approximation of the *Personalized PageRank* (APPR) vector if the start distribution vector s is sparse. The idea behind the *push* algorithm is only to consider a node in the local neighborhood if the probability to visit the node is significantly larger than the probability to visit any other node from the rest of the graph. This leads to a sparse solution meaning that only relatively few nodes of the underlying graph are contained in the resulting APPR vector.

Algorithm 1 describes the computation of APPR using a variant of the *push* operation on lazy random walk transition matrices of undirected unweighted graphs. This algorithm was proposed in [3], where APPR is used to partition graphs. We describe an

Algorithm 1 ApproximatePPR

```
Input: Starting vector $s$, Teleportation probability $\alpha$, Approximation threshold $\epsilon$
Output: APPR vector $p$
1: $p = \vec{0}, r = s$
2: while $r(u) \geq \epsilon d(u)$ for some vertex $u$ do
3:     pick any $u$ where $r(u) \geq \epsilon d(u)$
4:     push(u)
5: end while
6: return $p$
```

Algorithm 2 push

```
Input: Vertex $u$
1: $p(u) = p(u) + (2\alpha/(1+\alpha))r(u)$
2: for $v$ with $(u,v) \in E$ do
3:     $r(v) = r(v) + ((1-\alpha)/(1+\alpha))r(u)/d(u)$
4: end for
5: $r(u) = 0$
```

adapted version from [36] which converges faster. The algorithm maintains two vectors: the solution vector p and a residual vector r. The vector p is the current approximation of the PPR vector and vector r contains the approximation error or the not yet distributed probability mass. $p(u)$ and $r(u)$ are the entries in vectors p and r corresponding to node u, $d(u)$ is the degree of node u. In each iteration the algorithm selects a node with sufficient probability mass in vector r. This probability mass is spread between the node entry in p and the entries of its direct neighbors in r. In each step, the exact PPR is the linear combination of the current solution vector p and the PPR solution for r, i.e., $pr(s) = p + pr(r)$. The algorithm can also be trivially adapted to directed graphs and graphs with weighted edges.

3.2.1 Local Label Distribution. In our approach we first compute the APPR vector for each node. Before APPR is computed for node v, the corresponding entry $s(v)$ in starting vector s is set to one and all other entries to zero. Therefore, the APPR vector of v describes the importance of local neighbors only from its point of view.

In the APPR result matrix **APPR**, each row corresponds to the APPR vector of the corresponding node. The local label distribution representation $\mathbf{X} \in \mathbb{R}^{n \times l}$ is computed by manipulating **APPR** such that the diagonal is set to zero to exlude information about the own labels and then multiplying the resulting matrix $\widehat{\mathbf{APPR}}$ with the label matrix $\mathbf{Y}_{train}$. The entry $\mathbf{X}_{v\, y_j}$ can be interpreted as the probability that a random walk starting from node v stops at a neighbor with label y_j.

The local label distribution can be used as a node embedding vector which can be passed into an arbitrary classification algorithm. In our experiments, we employ a multi-layer perceptron with three layers, i.e., an input layer taking the local label distribution matrix $\mathbf{X}$ as input, a dense hidden layer with 16 units and an ReLU activation, and finally a dense layer retrieving the output. Formally, the hidden layer H can be described as

$$H = ReLU(\widehat{\mathbf{APPR}} \cdot \mathbf{Y}_{train} \cdot \mathbf{W}_1)$$

with $\mathbf{W}_1$ denoting the weight matrix. Note that the bias is omitted for the sake of better readability.

4 RELATED WORK

Numerous approaches for semi-supervised learning on graphs have been proposed recently. These can be categorized into unsupervised node embedding techniques and semi-supervised techniques.

4.1 Unsupervised Node Embedding

Lots of recent developments related to learning from structural relationships have focussed on learning *node embeddings*, where a latent vector representation is learned for each node, reflecting its connectivity in the underlying graph. The learned node embeddings can be used as an input to a subsequent down-stream task, such as node classification. Random walk based methods [16, 32] sample a number of random walks from the graph and nodes are related if they have common neighbors. *LINE* [37] is another variant, which considers direct first- and second-order proximities instead of random walks. *Graph2Gauss* [7] learns similarity to hop neighborhoods and embeds each node as a Gaussian distribution to allow for uncertainty in the representation. *GECS* [2] uses connections subgraphs to determine appropriate node neighborhood. More closely related to our approach, *LASAGNE* [13] relies on APPR to determine relevant context nodes. Other works perform matrix factorization. For instance, *GraRep* [9] factorizes a sequence of k-step log-probability matrices with SVD and concatenates the resulting low-dimensional node representations to form the final representations. Abu-El-Haija et al. propose matrix factorization of random-walk occurrence matrix with different approaches to determine context window size distribution [1]. *SDNE* [40] uses a multi-layer auto-encoder model to capture non-linear structures based on direct first- and second-order proximities. Authors of [10] propose embeddings in hyperbolic space. *HARP* [11] addresses the local minima problem and introduce an iterative scheme for learning of node representations which can be used with different embedding learning methods. An input graph is coarsened on different levels and node representations are learned starting with the coarsest graph and learned embeddings are provided as initializations for the embeddings of subsequent finer graphs. While the above methods rely on the homophily assumption, *struc2vec* [33] aims at learning representations which relate structurally similar nodes instead of nodes which are close in the graph. It does so by using degree sequences in neighborhoods of different sizes. All of the above approaches are *transductive* in the sense that labels can only be predicted for unlabeled nodes observed already at training time. The *GraphSAGE* [17] framework introduces *inductive* node embeddings. The basic idea is to learn an embedding function by sampling and aggregating node attributes in local neighborhoods. The embedding function can further be learned with a supervised loss function. Inductive models are also obtained by considering node attributes. *Variational Graph Auto-Encoders* [21] learn node representations using a variational auto-encoder, where the encoder is a two-layer GCN. The model can be applied to attributed and non-attributed graphs.

4.2 Semi-Supervised Learning on Graphs

Compared to separately optimizing steps in a semi-supervised learning pipeline, as is the case for semi-supervised learning with pre-trained node embeddings, end-to-end training usually leads to better performance on the supervised learning objective.

One direction is *Laplacian Regularization*, where the prediction loss is augmented with an unsupervised loss function based on the graph's Laplacian matrix, encoding the homophily assumption that close-by nodes should have the same label. Related approaches include *Manifold Regularization* [5], a kernel-based method, and *Deep Semi-Supervised Embedding* [41] which incorporates node embeddings by augmenting neural network models with an embedding layer. Both of these methods generalize to attributed graphs. The *ICA* algorithm [30] starts with the observed labels and iteratively classifies unlabeled nodes based on aggregated node attributes in local neighborhoods. At the end of each phase, the nodes classified with highest certainty are added to the ground truth for the next phase. *Label Diffusion* methods [15, 19, 44, 45] are more closely related to our work. Similarly to our method they create embeddings based on labels in local neighborhoods. The basic idea is based on mincuts [15] and labels are inferred based on majority vote. Therefore, Label Diffusion approaches do not exploit the effect of similar label distributions in a graph. More recent methods, as proposed in [29] and [43], also classify nodes based on labels in local neighborhoods. They learn a model which predicts node labels from a feature vector describing the local k-neighborhood. Both methods assume unattributed graphs.

Instead of imposing regularization, *Planetoid* [42] combines the prediction loss with node embeddings by training a joint model which predicts class labels as well as graph context for a given node. The graph context sampled from random walks as well as the set of nodes with shared labels. This allows Planetoid to relate nodes with similar labels even if they are not close in the graph. Thus, Planetoid does not rely on a strong homophily assumption. In addition to a connectivity-based variant, *Planetoid-G*, the authors propose two further architectures, which incorporate node attributes. The transductive variant *Planetoid-T* starts with pre-trained embeddings and alternately optimizes the prediction and embedding loss functions. The inductive variant *Planetoid-I* on the other hand predicts the graph context from the node features instead.

Another important direction which has recently gained increasing attention is concerned with generalizing deep neural network architectures to graph-structured domains. As the general approach consists of incorporating graph structure into supervised learning, these models assume an attributed graph. However, they can naturally be applied to non-attributed graphs by using the identity matrix as the attribute matrix. The vast majority of neural network based models for semi-supervised learning on graphs can be described within a message-passing framework. In a *Message Passing Neural Network (MPNN)* [14], each node has a hidden state which is updated iteratively during training. The initial hidden state of a node corresponds to its attribute vector. In a first step, messages from v_i's neighborhood are received and aggregated, where a message from neighbor v_j depends on v_i's and v_j's hidden states. In a second step, v_i's state is updated by combining it with the aggregated messages. An important special case are *Graph Convolution Networks* [8, 12, 20, 23, 26, 27, 39] which aggregate node attributes over local neighborhoods with spatially localized filters, similar to classical convolutional networks on images [22]. The *ChebNet* [12] aggregates messages from neighbors analogously to the eigenvectors of the graph's Laplacian matrix. The update function ignores the previous state and applies a non-linear activation. The resulting filters are k-localized. The *GCN* [20] is a simplification of the ChebNet, which only considers one-hop neighbors. Messages are aggregated according to a normalized adjacency matrix. In the update phase, the aggregated messages are multiplied with a learned filter matrix with a ReLU activation. For graph convolution networks, the number of message passing iterations corresponds to the number of layers.

5 EVALUATION

We evaluate our approach by performing node-label prediction and compare the quality in terms of micro F_1 score for multiclass prediction tasks, respectively micro F_1 and macro F_1 scores for multilabel prediction tasks, against state-of-the-art methods.

For both tasks, we compare our model against the following approaches:

- *Adj*: a baseline approach which learns node embeddings only based on the information contained in the adjacency matrix
- *GCN*$_1$*_only_L*: a GCN which applies convolution on label matrix Y. We use one convolution layer on the adjacency matrix without self-links, followed by a dense output layer[1]
- *noFeat GCN*$_2$: the standard 2-layer GCN as published by Kipf et al. [20] without using the node attributes
- *DeepWalk*: the DeepWalk model as proposed in [32]
- *node2vec*: the node2vec model as proposed in [16]
- *Planetoid-G*: the Planetoid variant which does not use attribute information [42] [2]

Our model is denoted as *LD* (short for Label Distribution). For these experiments we train a simple feed-forward neural network which takes the label distribution based representations as input and retrieves class probabilities as output.

Note that we omit the comparison to label propagation [45] since Yang et al. already showed that the *Planetoid* model outperforms this approach [42].

5.1 Multiclass Prediciton

5.1.1 Experimental Setup. For the multiclass label prediciton task we use the following three text classification benchmark graph datasets [28, 35]:

- Cora. The Cora dataset contains 2'708 publications from seven categories in the area of ML. The citation graph consists of 2'708 nodes, 5'278 edges, 1'433 attributes and 7 classes.
- CiteSeer. The CiteSeer dataset contains 3'264 publications from six categories in the area of CS. The citation graph consists of 3'264 nodes, 4'536 edges, 3'703 attributes and 6 classes.
- Pubmed. The Pubmed dataset contains 19'717 publications which are related to diabetes and categorized into 3 classes. The citation graph consists of 19'717 nodes, 44'324 edges, 500 attributes and 3 classes.

For each graph, documents are denoted as nodes and undirected links between documents represent citation relationships. If node attributes are applied, bag-of-words representations are used as attribute vectors for each document.

We split the data as suggested in [42], i.e., for labeled data our training sets contain 20 randomly selected instances per class, the test sets consist of 1'000 instances, and the validation sets contain 500 instances for each method. The remaining instances are used as unlabeled data. For comparison we use the prediction micro F_1 scores which we collected over 10 different data splits.

Since the numbers of iterations for sampling the graph contexts and the label contexts for *Planetoid* are suggested only for the *CiteSeer* data set, we adapted these values relative to the number of nodes for each graph. For *node2vec*, we perform grid searches over the hyperparameters p and q with $p, q \in \{0.25, 0.5, 1.0, 2.0, 4.0\}$ and use window size 10 as proposed by the authors. For all models except *Planetoid* unless otherwise noted, we use one hidden layer with 16 neurons and regularization, learning rate and training procedure as in [20]. Considering our model, we use $\alpha \in \{0.1, 0.2, \ldots, 0.9\}$ as values for the teleportation parameter and $\epsilon = 1e^{-5}$ as approximation threshold to compute the APPR vectors for each node.

We present results computed on the test sets for the best performing hyperparameters. The best performing hyperparameters for all models are determined by using the validation sets.

5.1.2 Results. Figure 2 shows boxplots depicting the micro F_1 scores we achieved for the multiclass prediction task for each considered model on the *Cora*, *CiteSeer* and *Pubmed* networks.

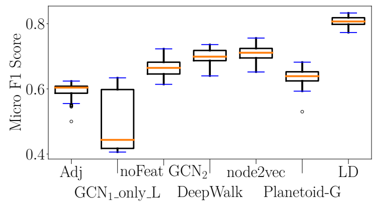
The baseline approach *GCN*$_1$*_only_L*, i.e., the one layer GCN model which only uses the label distributions of the neighboring nodes to predict a node's label, shows worst results among the considered models. However, these scores are still promising that the labels may improve the task of learning "good" representations. The baseline method which considers the corresponding rows of the adjacency matrix as node representations, i.e., *Adj*, achieves slightly better results for all three datasets. For the *GCN* and *Planetoid* models that do not make recourse to attribute information, i.e., *noFeat GCN*$_2$, resp. *Planetoid-G*, the retrieved micro F_1 values are slightly lower than the ones achieved by *DeepWalk* and *node2vec*. Our model improves the results produced by *node2vec*, which means that the label distributions are indeed a useful source of information, although the baseline *GCN*$_1$*_only_L* shows, especially for *Pubmed*, rather poor results. This may be reasoned by the fact that this model only considers the label distribution of a very local neighborhood (in fact one hop neighbors). However, collecting the label distribution from a more spacious neighborhood gives a significant boost in terms of prediction accuracy. Indeed the best results for the *LD* approach are reached for $\alpha = 0.1$, which corresponds to a rather spacious neighborhood exploration.

5.2 Multilabel Classification

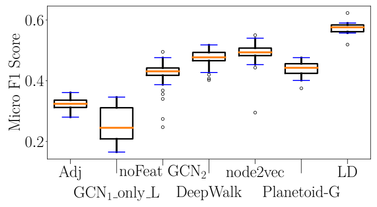
5.2.1 Experimental Setup. We also perform multilabel node classifications on the following two multilabel networks:

- BlogCatalog [38]. This is a social network graph where each of the 10,312 nodes corresponds to a user and the 333,983 edges represent the friendship relationships between bloggers. 39 different interest groups provide the labels.
- IMDb Germany. This dataset is taken from [13]. It consists of 32,732 nodes, 1,175,364 edges and 27 labels. Each node represents an actor/actress who played in a German movie. Edges connect actors/actresses that were in a cast together and the node labels represent the genres that the corresponding actor/actress played.

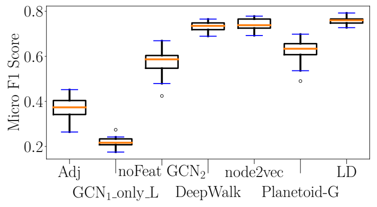
[1]See 3.1 for the explanation why only one convolution layer makes sense
[2]Unless stated differently we use for all competitors the parameter settings as suggested by the corresponding authors. Except for minor adaptations, e.g., to include label information in the one layer GCN models or to make the Planetoid models applicable for multilabel prediciton tasks, we use the original implementations as published by the correpsonding authors.



(a) Micro F_1 scores for *Cora*.

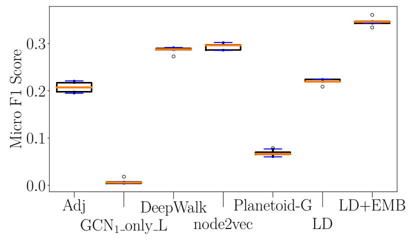


(b) Micro F_1 scores for *CiteSeer*.

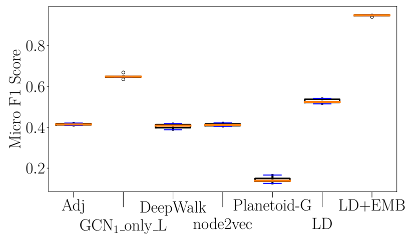


(c) Micro F_1 scores for *Pubmed*.

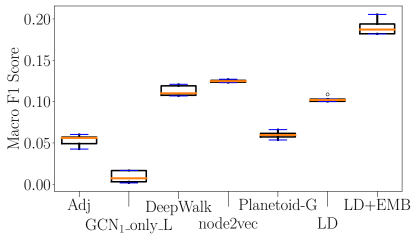
Figure 2: Micro F_1 scores for the three benchmark data sets.



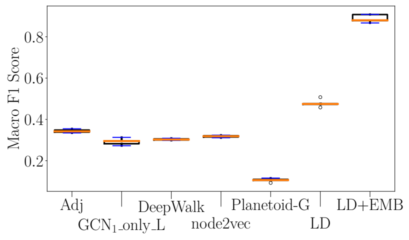
(a) Micro F_1 scores for *BlogCatalog*.



(a) Micro F_1 scores for *IMDb Germany*.



(b) Macro F_1 scores for *BlogCatalog*.



(b) Macro F_1 scores for *IMDb Germany*.

Figure 3: Micro F_1 and macro F_1 for *BlogCatalog*.

Figure 4: Micro F_1 and macro F_1 for *IMDb Germany*.

Since the fraction of positive instances is relatively small for most of the classes, we use weighted cross-entropy as loss function. Therefore, the loss caused by erroneously classified positive instances is weighted higher. We use weight 10 in all our experiments. For the same reason we report micro F_1 and macro F_1 score metrics to measure the quality of the considered methods. We compare our model to the featureless models that we already used for the multiclass experiments [3].

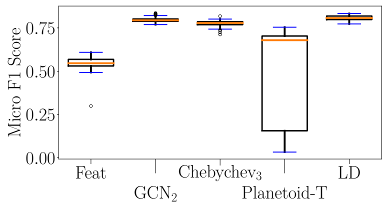
We split the data into training, validation and test set so that 70% of all nodes were used for training, 10% for validation and 20% of the data were used to test the model. Note that we could not use stratified sampling splits for these experiments since we optimize for all classes simultaneously instead of using one-vs-rest classifiers [4]. The hyperparameter setting is as described above. For this set of experiments we ran each model, except for *Planetoid-G*, 10 times on five different data splits. Due to the long runtime of *Planetoid-G* we trained this model only three times on two data splits.

5.2.2 Results. The results for the *BlogCatalog* graph are shown in Figure 3. For this network, only using the label information from the direct neighborhood of a node is not useful to infer its labels, c.f., *GCN_1_only_L*. However, incorporating the label distribution of somewhat larger neighborhoods as for our model (again, we also use the APPR matrix calculated for small values of α to determine the label distribution in neighborhoods that span more than 1-hop neighbors) seems to improve the results for the prediciton task significantly. In fact, our model achieves similar, but slightly worse performance than *node2vec* and *DeepWalk*. Given these results, we also combined the node embeddings based on local label distributions with embeddings that capture structural properties. To capture the structural properties we select a very simple approach: we multiply an embedding matrix with the preprocessed adjacency matrix as in Kipf et al. [20]. The embedding matrix is randomly initialized. Note that the structural similarity is defined via direct neighbors. The resulting representation is concatenated with the hidden layer of the *LD* model and the rest of the *LD* model remains the same. The embedding weights are learned jointly with the rest of the model. The hidden layer H for the resulting model, denoted as *LD+EMB*, can be formalized as

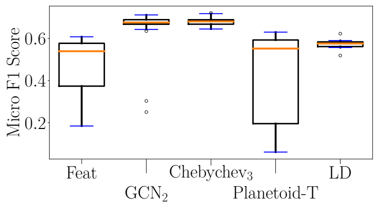
$$H = ReLU\left(\left[\mathbf{E}\hat{\mathbf{A}}, \widehat{\mathbf{APPR}} \cdot \mathbf{Y}_{train} \cdot \mathbf{W}_1\right]\right),$$

[3]To adapt the *Planetoid-G* implementation for multilabel classification, we use a *sigmoid* activation function at the output layer and also slightly changed the embedding learning step. Entities that are used as context and have the same labels as the node itself are sampled from all classes to which the node belongs to.

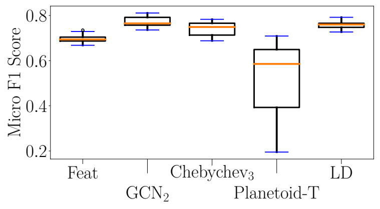
[4]That is why our results for *node2vec* and *DeepWalk* on the BlogCatalog network are slightly worse than reported in [16]



(a) Micro F_1 scores for *Cora*.



(b) Micro F_1 scores for *CiteSeer*.



(c) Micro F_1 scores for *Pubmed*.

Figure 5: Comparison against attribute-based methods: micro F_1 scores for the three benchmark data sets.

with $[\cdot, \cdot]$ denoting the concatenation operation, $\mathbf{E}$ being the embedding matrix and $\hat{\mathbf{A}}$ being the preprocessed adjacency matrix as in [20]. Again, the bias is omitted for better readability. Having a look at the scores for the *LD+EMB* model, this combination further improves the outcome of the prediction.

For the *IMDb Germany* network, for which the results can be seen in Figure 4, the labels of even very local neighborhoods are already very expressive. Recalling how this network is constructed, we can expect the latter fact and also the superior performance of our model over the two random walk based methods. Particularly noteworthy for this network is the gain of accuracy that the combination of information from both sources, label distribution and structural properties, achieves.

5.3 Comparison to Attribute-Based Methods

To show the power of incorporating label information into the generation process for node embeddings, we also compare our model against the following state-of-the-art attribute-based methods:

- *Feat*: a baseline approach which predicts node labels only based on the node attributes without considering the underlying graph structure (borrowed from [42])
- *GCN*$_2$: the standard 2-layer GCN as published in [20]
- *Chebychev*$_3$: the spectral convolution method which uses chebychev filters as presented in [12]; as in [20] we also use 3rd order chebychev filters
- *Planetoid-T*: the semi-supervised Planetoid framework which uses attribute information as proposed in [42]

For this set of experiments, we again perform multiclass prediciton on the three benchmark text classification datasets and report the prediction accuracy in terms of micro F_1 scores to measure the quality of the retrieved node representations. Note that in contrast to the competitors, our model still does not make use of the node attribute information. The results are depicted in Figure 5 and clearly show that our model can definitely compete with the attribute-based methods and hence is a powerful alternative in cases when no node attributes are present.

5.4 Impact of the α Parameter

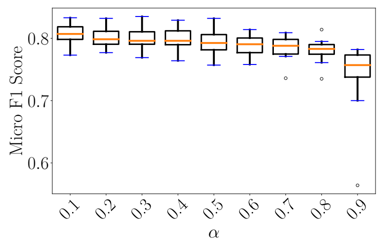
Figure 6 depicts the micro F_1 scores achieved for different values of the teleportation parameter α on the three benchmark datasets. As can be seen, particularly for the *Pubmed* network, the model is quite sensitive to the choice of this parameter. Recall that the teleportation parameter determines how far the neighborhood of each node shall be taken into consideration to get the label distributions for each node. Therefore it might make sense to set the α parameter to a small value so that more labels are collected which in turn leads to a more accurate estimation of the local label distribution. On the other hand, this may not hold in every scenario, for instance if the distribution of classes is heterogeneous, i.e., some classes may only appear in areas of the graph where classes are concentrated locally, while other classes may appear in areas where many classes are mixed even within local neighborhoods. An interesting direction for future work is therefore to optimize for some "good" α value in a data-driven manner. This may be done either by pre-defining a set of different values of α and approaching for the best of these, or by trying to optimize for some "good" α value during the learning procedure. Also, the underlying task, e.g., node classification, may benefit from finding "good" values of α for each node individually rather than relying on a global solution.

6 CONCLUSION

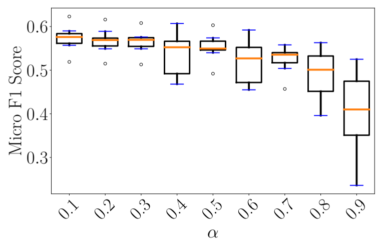
In this paper, we have introduced a novel label-based approach for semi-supervised node classification. In particular, our method aggregates labels from local neighborhoods using APPR. Most existing approaches consider nodes to be similar, if they are closely related in the graph. Methods for attributed graphs additionally take attributes of the neighboring nodes into account. In contrast, our method can relate nodes even if they are not close-by in the graph and makes more effective use of the labels provided for training to improve the classification quality for graphs with and without node attributes. It is further applicable to nodes unseen during training. The results of our experiments on various real-work datasets demonstrate that local label distributions are able to significantly improve classification results in the multiclass and multilabel setting. Our model is even competitive with state-of-the-art models, which take node attributes into consideration. In a first experiment on multilabel datasets, we were already able to significantly boost the performance by using a simple combination of our model with node embeddings.

For future work, we plan to address the problem of selecting a suitable teleportation parameter α. The α parameter controls the extend of the considered local neighborhood and often has a significant impact on the prediction quality. Performing a grid search to determine a good parameter value is a time consuming task. Furthermore, for different classes varying teleportation parameters might yield the best results.

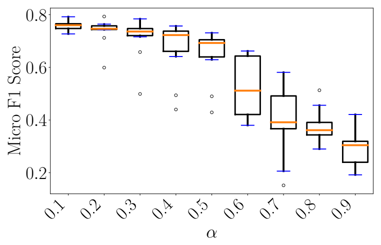
We also aim at further improving the prediction accuracy by further investigating how to effectively combine label-based features with different other kinds of features, such as node attributes, edge attributes or node embeddings in a semi-supervised model. Our



(a) Micro F_1 scores for *Cora.*



(b) Micro F_1 scores for *CiteSeer.*



(c) Micro F_1 scores for *Pubmed.*

Figure 6: Micro F_1 scores for the three benchmark data sets when considering different locality levels for node neighborhoods.

approach could also be extended to solve additional graph learning tasks, such as link prediction or identification of nodes with unexpected labels for detecting labeling errors or outlier nodes.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alex Alemi. 2017. Watch your step: Learning graph embeddings through attention. *arXiv preprint arXiv:1710.09599* (2017).
[2] Saba A Al-Sayouri, Pravallika Devineni, Sarah S Lam, Evangelos E Papalexakis, and Danai Koutra. 2016. GECS: Graph Embedding Using Connection Subgraphs. (2016).
[3] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *Proc. of IEEE FOCS.* IEEE, 475–486.
[4] James Atwood and Don Towsley. 2015. Search-Convolutional Neural Networks. *CoRR* abs/1511.02136 (2015). arXiv:1511.02136 http://arxiv.org/abs/1511.02136
[5] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research* 7, Nov (2006), 2399–2434.
[6] Pavel Berkhin. 2006. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics* 3, 1 (2006), 41–62.
[7] Aleksandar Bojchevski and Stephan Günnemann. 2017. Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815* (2017).
[8] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral Networks and Locally Connected Networks on Graphs. *CoRR* abs/1312.6203 (2013). http://arxiv.org/abs/1312.6203
[9] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proc. of CIKM.* ACM, 891–900.
[10] Benjamin Paul Chamberlain, James Clough, and Marc Peter Deisenroth. 2017. Neural Embeddings of Graphs in Hyperbolic Space. *arXiv preprint arXiv:1705.10359* (2017).
[11] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. 2017. HARP: Hierarchical Representation Learning for Networks. *arXiv preprint arXiv:1706.07845* (2017).
[12] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Eds.). 3844–3852.
[13] Evgeniy Faerman, Felix Borutta, Kimon Fountoulakis, and Michael W Mahoney. 2017. LASAGNE: Locality And Structure Aware Graph Node Embedding. *arXiv preprint arXiv:1710.06520* (2017).
[14] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212* (2017).
[15] David F Gleich and Michael W Mahoney. 2015. Using local spectral methods to robustify graph-based learning algorithms. In *Proc. of the ACM SIGKDD.* 359–368.
[16] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proc. of ACM SIGKDD.* 855–864.
[17] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS.* 1025–1035.
[18] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *Proc. of the 12th WWW.* ACM, 271–279.
[19] Thorsten Joachims. 2003. Transductive learning via spectral graph partitioning. In *Proc. of ICML.* 290–297.
[20] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[21] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).
[22] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551.
[23] Ron Levie, Federico Monti, Xavier Bresson, and Michael M. Bronstein. 2017. CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters. *CoRR* abs/1705.07664 (2017). arXiv:1705.07664
[24] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated Graph Sequence Neural Networks. In *ICLR.*
[25] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: Homophily in social networks. *Annual review of sociology* 27, 1 (2001), 415–444.
[26] Yann LeCun Mikael Henaff, Joan Bruna. 2015. Deep Convolutional Networks on Graph-Structured Data. *arXiv preprint arXiv:1506.05163* (2015).
[27] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. 2016. Geometric deep learning on graphs and manifolds using mixture model CNNs. *CoRR* abs/1611.08402 (2016). arXiv:1611.08402 http://arxiv.org/abs/1611.08402
[28] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. 2012. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs.*
[29] Sharad Nandanwar and M Narasimha Murty. 2016. Structural neighborhood based classification of nodes in a network. In *Proc. of ACM SIGKDD.* 1085–1094.
[30] Jennifer Neville and David Jensen. 2000. Iterative classification in relational data. In *Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data.* 13–20.
[31] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: bringing order to the web. (1999).
[32] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proc. of ACM SIGKDD.* 701–710.
[33] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *Proc. of ACM SIGKDD.* ACM, 385–394.
[34] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80.
[35] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93.
[36] Julian Shun, Farbod Roosta-Khorasani, Kimon Fountoulakis, and Michael W. Mahoney. 2016. Parallel Local Graph Clustering. *Proc. VLDB Endow.* 9, 12 (Aug. 2016), 1041–1052.
[37] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proc. of WWW.* ACM, 1067–1077.
[38] Lei Tang and Huan Liu. 2009. Relational learning via latent social dimensions. In *Proc. of ACM SIGKDD.* ACM, 817–826.
[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2017. Graph Attention Networks. *arXiv preprint arXiv:1710.10903* (2017).
[40] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proc. of ACM SIGKDD.* ACM, 1225–1234.
[41] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. 2012. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade.* Springer, 639–655.
[42] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *Proc. of ICDM.* 40–48.
[43] Wei Ye, Linfei Zhou, Dominik Mautz, Claudia Plant, and Christian Böhm. 2017. Learning from Labeled and Unlabeled Vertices in Networks. In *Proc. of ACM SIGKDD.* ACM, 1265–1274.
[44] Denny Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. 2004. Learning with local and global consistency. In *Advances in neural information processing systems.* 321–328.
[45] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *Proc. of ICML.* 912–919.

2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT) | 978-1-6654-1924-6/20/$31.00 ©2020 IEEE | DOI: 10.1109/WIIAT50758.2020.00009

Ada-LLD: Adaptive Node Similarity Using Multi-Scale Local Label Distributions

Evgeniy Faerman, Felix Borutta, Julian Busch, Matthias Schubert
Ludwig-Maximilians-Universität München
{faerman, borutta, busch, schubert}@dbs.ifi.lmu.de

***Abstract*—In many applications, data is represented as a network connecting nodes of various types. While types might be known for some nodes in the network, the type of a newly added node is typically unknown. In this paper, we focus on predicting the types of these new nodes based on their connectivity to the already labeled nodes. To tackle this problem, we propose *Adaptive Node Similarity Using Multi-Scale Local Label Distributions (Ada-LLD)* which learns the dependency of a node's class label from the distribution of class labels in this node's local neighborhood. In contrast to previous approaches, our approach is able to *learn* how class labels correlate with labels in variously sized neighborhoods. We propose a neural network architecture that combines information from differently sized neighborhoods allowing for the detection of correlations on multiple scales. Our evaluations demonstrate that our method significantly improves prediction quality on real world data sets. In the spirit of reproducible research we make our code available[1].**

I. Introduction

In many applications of node classification in graphs, we encounter a semi-supervised setting in the way that the labels for all nodes in an old state of the network are known and we need to predict the labels of nodes freshly connected to the network. For example, consider a career network where users are labeled by potential job interests. Now, when a new user enters the network, she/he would link to already known users and we are interested to make a predictions on job interests based on this connectivity. Because many real-life graphs are characterized by high variations of patterns for the same labels across the same graph [24], methods having a fixed semantic interpretation of linked nodes often fail to model all relevant relationships. For example, a head hunter would link to companies and professionals but rather not to other head hunters, whereas scientists often mostly link to other scientists. However, such heterophily (see Fig. 1b), or homophily (see Fig. 1a) assumptions, respectively, are rather strict and in many applications not realistic, as for instance scientists may also have other relationships than those to other scientists. Moreover, such correlation patterns do not have to be fixed for entities of the same type and over the entire network (see Fig. 1c). For example, scientist at companies most likely have a larger portion of business managers within their neighborhoods than scientists from a university, although they may reside in the same network and have the same types. In this paper, we argue that purely homophily or heterophily-based methods

[1]https://github.com/adalld/wi2020

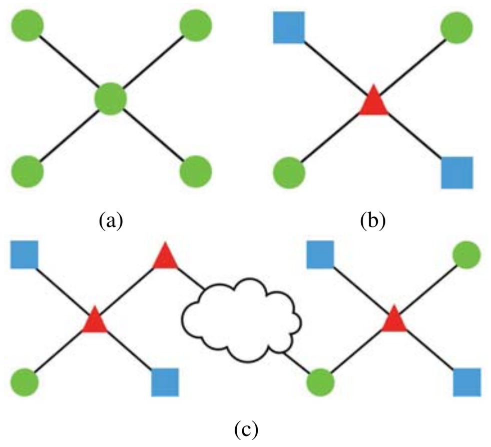


Figure 1: Examples for Homophily (1a), Heterophily (1b) and a combination of different label correlation patterns for different instances of the same class (1c).

are limited when predicting node types based on complex connection relationships. Therefore, our new method Adaptive Node Similarity Using Multi-Scale Local Label Distributions (Ada-LLD) aims at capturing various relationships between the types in different levels of node neighborhoods. In particular, our method uses Personalized Page Rank to describe the local neighborhood of a node. Since a meaningful extension of this local neighborhood often depends on the type of relationship, we consider multiple neighborhoods each having a larger extension then the previous one. After computing the label distribution in each neighborhood, we propose trainable functions to combine the local label distributions in each neighborhood into a meaningful node description for classification. For parameter optimization, we formulate these functions in conjunction with the node classifier as neural network and optimize them jointly using stochastic gradient descent. Since Ada-LDD works on local neighborhoods it is not required that the complete network is connected as required for label propagation methods. In general, our method is applicable to all nodes having a sufficient connectivity to already labeled nodes which is a common setting in incrementally extended networks. In our experiments, we show that Ada-LLD can outperform various sate-of-the-art node classifiers on various real-world graph data sets.

II. Related Work

Numerous approaches related to semi-supervised node classification have been proposed recently. These can be categorized into unsupervised and semi-supervised node embedding techniques. Unsupervised node embedding techniques learn a latent vector representations for each node in a graph. Homophilic methods [3], [5], [7], [10], [11], [25], [28], [30], [33], [35] relate nodes if they are close-by in the graph. These methods mainly differ w.r.t. how node neighborhoods are defined, e.g., based on different random walk strategies or fixed k-hop distances. Compared to separate optimizing steps in a semi-supervised learning pipeline, as is the case for semi-supervised learning with pre-trained unsupervised node embeddings, end-to-end training usually leads to better performance on the supervised learning objective.

Label Diffusion [23], [32], [36]–[38] methods propagate labels from labeled nodes until convergence and classify based on majority vote. While classical methods [36]–[38] rely on the homophily assumption, *Cosine Label Propagation* [23] applies label propagation to a similarity graph constructed from the cosine similarity of the nodes' adjacency vectors. The authors further propose *2-Step Label Propagation* [23] that skips immediate neighbors during label propagation. *Dynamic Label Propagation* [32] extends label propagation by inducing label correlation between nodes into the transition matrix to reinforce propagation between nodes with similar label distributions. In contrast to our approach, these label distributions consider a node's own labels and not labels occurring within the node's neighborhood. The *Belief Propagation* [8], [15], [22] approaches also perform label propagation and additionally are able to model various label relationship patterns like homophily, heterophily or mixed patterns. However, these methods explicitly expect a parameterized transition matrix that describes relationships between labels. This limits their applicability, as such relationships might be rather complex. Note, that our approach is able to *learn* these relationships.

Similarly to node embeddings, all of the above methods are not able to make use of information in distant or disconnected parts of a graph. Furthermore, graph diffusion methods do not have separate training and inference steps and if the graph changes, these methods have to be executed again. *ICA* and *GS* algorithms [26] first train a relational classifier, which makes the predictions based on labels of direct neighbors. The same fixed classifier is iteratively used to assign labels to unlabeled nodes and nodes labeled in previous iterations are utilized for the classifications of the neighbors. The *Planetoid* model [34] combines the prediction loss with node embeddings by training a joint model that predicts class labels and graph context for a given node. It samples context nodes from the local neighborhood as well as possibly distant nodes with a shared label.

A different but important direction is semi-supervised learning on attributed graphs. The vast majority of recently proposed neural network based models can be described within *Message Passing Neural Network (MPNN)* frameworks [9]. A special case are *Graph Convolution Networks (GCN)* [1], [4], [6], [14], [16]–[18], [20], [31] which aggregate node attributes over local neighborhoods with spatially localized filters, similar to classical convolutional networks on image data. Thus, these methods are not able to learn direct correlations between a node's label and labels of neighboring nodes, be it homophily or a general type of correlation. Similarly to our approach, one might attempt to use MPNN in combination with class labels instead of attributes. This would require to ensure that a node will not at some point receive a message containing information about its own class label. However, this will happen after a sufficient number of message passing iterations if a node is reachable from itself. This problem is non-trivial since the MPNN framework requires the whole feature matrix for the forward pass and computes updates based on all labeled nodes in the training set. A naive approach of masking the nodes' own labels would result in a different feature matrix for each node and therefore lead to infeasible training costs.

A. Comparison

We compare related methods for node classification w.r.t. several key properties which describe different aspects of how nodes can be related with each other for classification. In addition to the discussion below, an overview is provided in Table I.

Homophily. Whether the method is able to model homophily. MPNN and Ada-LLD are able to learn homophily, but are not doing so explicitly. MPNN however is only able to model homophily indirectly via additional node attributes. The remaining methods can either be parametrized explicitly to model homophily or focus on homophily by design.

Heterophily. Whether the method is able to model heterophily. MPNN and Ada-LLD are able to model heterophily, but are not doing so explicitly. Again, MPNN is only able to model heterophily indirectly via additional node attributes. While Belief Propagation needs to be parametrized explicitly to model heterophily, advanced Label Diffusion methods focus on heterophily by design.

Adaptive. Whether the model is able to adaptively learn appropriate node similarities without explicitly relying on either homophily or heterophily. MPNN learns general correlations only between a node's label and additional node attributes appearing in the neighborhood. Ada-LLD learns such correlations directly from the neighboring node labels. The other methods are not adaptive.

Local Variation. Whether the method is able to model homophily or different heterophily patterns for the same label at the same time. The only methods capable of modeling this property are MPNN and Ada-LLD, where MPNN again relies on additional node attributes.

Labels. Whether node similarity is modeled directly based on labels. Homophilic graph embeddings are unsupervised and only consider graph topology. MPNN models node similarity based on node attributes. Planetoid considers labels directly only via sampling of nodes with the same labels. Label Diffusion and Belief Propagation consider labels directly by

Table I: Comparison to related node classification methods based on whether they fulfill (✓) the desired key properties or not (✗). Parentheses indicate partial fulfillment.

Method	Homophily	Heterophily	Adaptive	Local Variation	Labels	Remote
Homophilic Node Embeddings [3], [5], [7], [10], [19], [25], [28], [33], [35]	✓	✗	✗	✗	✗	✗
Label Diffusion [23], [32], [36]–[38]	✓	✓	✗	✗	✓	✗
Belief Propagation [8], [15], [22]	✓	✓	✗	✗	✓	✗
Planetoid [34]	✓	✗	✗	✗	(✓)	(✓)
MPNN [9]	(✓)	(✓)	(✓)	✓	✗	✓
Ada-LLD	✓	✓	✓	✓	✓	✓

diffusing them through the graph. Ada-LLD is the only method which directly *learns* general correlations between a node's label and neighboring labels.

Remote. Whether information from different parts, or disconnected components, respectively, of the graph can be incorporated to classify a node. MPNN and Ada-LLD are location-invariant and thus able to learn correlations based on the whole graph. Planetoid samples nodes with same class labels from different parts of the graph but only in addition to neighboring nodes. The remaining methods only take close-by nodes into account.

In conclusion, none of the existing methods fulfills all of the desired properties. Ada-LLD is the first method to support adaptive learning of direct label-based node similarity for node classification without a need of additional attributes.

III. Adaptive Node Similarity Using Local Label Distributions

Formally, our problem setting is *semi-supervised node classification* where we are given a graph $G = (V, E)$, represented by an adjacency matrix $A \in \mathbb{R}^{n \times n}$, and a label matrix $Y_{train} \in \{0,1\}^{n \times l}$ indicating class memberships for all nodes. The goal is to predict labels for a set of unlabeled nodes based on graph topology and already observed node labels. Unlabeled nodes correspond to constant zero rows in Y_{train}. To exclude trivial dependencies, we do not consider self-loops. Further, we consider two sub-settings: In *multiclass* classification, each node belongs to exactly one class. *Multilabel* classification refers to the general case in which each node may be assigned to one or more classes.

The Ada-LLD Model

The main idea of our approach is to learn general correlations between a node's label and the labels of neighboring nodes. According to this intuition, our core model predicts the label vector y_i for a given node v_i as

$$y_i = f\left(aggr\left(\{y_j \mid v_j \in \mathcal{N}(v_i)\}\right)\right),$$

where $\mathcal{N}(v_i)$ denotes the neighborhood of v_i, $aggr$ is an aggregation function and f is a classifier which predicts node labels based on the aggregated neighboring node labels.

A sensible choice for $aggr$ would be a weighted function which does not treat any labels equally but assigns more importance to labels of nodes which are more important to v_i. These considerations lead to *local label distributions* which are used as input features to our model.

For the classifier f, our default choice is a neural network with a single hidden layer H_1 for which we introduce two different alternatives. For prediction we use a fully connected layer

$$H_2 = q\left(W_{out} H_1 + b_{out}\right), \quad (1)$$

where $W_{out} \in \mathbb{R}^{h \times l}$ is the weight matrix, $b_{out} \in \mathbb{R}^{l}$ denotes the bias and q is the *softmax* activation $P(c_i) = {\exp(h_i)}/{\sum_{j=0}^{l} \exp(h_j)}$ in case of multi-class classification. If the classification problem is a multi-label one, class probabilities are computed using the *sigmoid* activation $P(c_i) = {1}/{(1+\exp(-h_i))}$. The resulting model is quite simple and efficient, yet sufficiently expressive to provide accurate predictions.

A. Local Label Distributions.

In many real-world graphs, most of the nodes can be reached within a few steps and often only a small set of neighboring nodes are important to a particular node. Considering a simple neighborhood of node n such as the k-hop neighborhood may lead to considering irrelevant nodes or missing important ones. In other words, nodes having the same shortest path distance to n might strongly vary in importance because only considering a single path often does not allow to distinguish the connectivity between nodes. In contrast, we propose to describe the neighborhood of node n by *Personalized PageRank (PPR)*. In other words, we compute the visiting probability of all nodes when starting a random walk at n and restart the random walk at n with a teleportation likelihood α. Thus, the smaller α is, the higher is the visiting probability of less connected nodes.

Local push-based algorithms [2], [13] can be used to compute *Approximate Personalized PageRank (APPR)* very efficiently for all nodes in a network and lead to sparse solutions where small, irrelevant entries (rarely visited nodes) are omitted. In particular, we consider the algorithm proposed in [27] as outlined in Algorithm 1. The algorithm requires two parameters to be set by the user. The teleportation parameter α determines the effective size of the neighborhood of a source node. The second parameter ϵ is a threshold which controls approximation quality and runtime. Given the PPR-vector π_i and the label matrix Y_{train}, we aggregate the neighboring

Algorithm 1 *Compute_LD*(v, α, ϵ)

```
Input: Source node $v$, Teleportation probability $\alpha$, Approximation threshold $\epsilon$, Label matrix $Y_{train}$
Output: Label distribution vector $ld$
1: // Compute APPR-vector for node $v$
2: $p = \vec{0}$, $r = \vec{0}$
3: $r(v) = 1$
4: while $r(u) \geq \epsilon d(u)$ for some vertex $u$ do
5:     pick any $u$ where $r(u) \geq \epsilon d(u)$
6:     $p(u) = p(u) + (2\alpha/(1+\alpha))r(u)$
7:     for $v$ with $(u, v) \in E$ do
8:         $r(v) = r(v) + ((1-\alpha)/(1+\alpha))r(u)/d(u)$
9:     end for
10:    $r(u) = 0$
11: end while
12: // Compute label distribution vector $ld$ for node $v$
13: $p(v) = 0$
14: $ld = pY_{train}$
15: return $ld$
```

labels of v_i as follows: $ld_i = \pi_i Y_{train} \in R^l$. The resulting vector ld_i is named *label distribution vector* of v_i. Intuitively, the entry $ld_{i,j}$ corresponds to the probability that a random walk starting at v_i stops at a node with label c_j.

B. Architecture

In scenarios as in the head hunter example, it might be useful to capture label information that occur on larger distance rather than being restricted to information given in the direct neighborhood, which in turn can be more advantageous e.g. in a social network. Moreover, it might happen that different nodes can be described best when considering the label information from differently sized neighborhoods rather than using the same scale for all nodes. To this end, we propose two architectures that combine label distributions in different extensions. Our first model, *LD_AVG*, tries to determine an optimal global combination of label distributions in different neighborhood extensions. Therefore, we combine input features by taking the weighted average, where there is a single weight per label distribution:

$$H_1 = q\left((\gamma_1 X_{\alpha_1} + \cdots + \gamma_k X_{\alpha_k}) W_{avg} + b_{avg}\right), \quad (2)$$

$X_{\alpha_i} \in \mathbb{R}^{n \times l}$ is label distribution matrix for single teleportation parameter and $[\gamma_1 \cdots \gamma_k]$ denotes the vector with learned parameters.

The *LD_AVG* model learns a single global weight for the whole neighborhood extension and is therefore not able to capture fine-granular dependencies between label distributions in different neighborhoods. However, the importance of neighboring labels can differ in the same neighborhood or there can be relevant combinations of labels from different neighborhoods. Therefore, for the more fine-granular combination of label neighborhoods we also evaluate the *LD_CONCAT* model which concatenates input features:

$$H_1 = q\left([X_{\alpha_1}, \cdots, X_{\alpha_k}] W_{concat} + b_{concat}\right), \quad (3)$$

As an alternative, the different label distribution representations are preprocessed by a linear layer, where each label distribution representation has its own preprocessing layer. The resulting features are concatenated in the hidden layer:

$$H_1 = q\left([X_{\alpha_1} W_1 + b_1, \cdots, X_{\alpha_k} W_k + b_k]\right), \quad (4)$$

For the last setting, one can additionally introduce an inductive bias by sharing weights in the preprocessing layer for different label neighborhoods:

$$H_1 = q\left([X_{\alpha_1} W_{shared} + b_{shared}, \cdots, X_{\alpha_k} W_{shared} + b_{shared}]\right). \quad (5)$$

The label distributions are combined in H_1 and fed to the fully connected prediction layer H_2 (cf., Eq. 1). The final MLP model is trained using *Stochastic Gradient Descent (SGD)* to minimize the *cross-entropy loss*:

$$\ell(v_i) = \sum_{j=1}^{l} -y_{i,j} \log P_{i,j},$$

where $P_{i,j}$ is the probability of class c_j for node v_i as predicted by our model.

C. Complexity

The main steps of our algorithm are summarized in Algorithm 2. The pre-computation as well as training steps are both highly efficient and scale to large graphs. Computing APPR for all k scales and all n source nodes requires $O(kn/\alpha\epsilon)$ operations [27]. Computing label distributions requires $O(km)$ operations on average where m is the average number of non-zero entries in an APPR-vector. Due to sparsity, it usually

Algorithm 2 *Ada-LLD*

```
Input: Graph $G = (V, E)$, Label matrix $Y_{train}$, Approximation threshold $\epsilon$, Teleportation parameters $\{\alpha_1, \ldots, \alpha_k\}$
Output: Trained classifier $f$
1: // Compute label distributions at each scale
2: declare $X \in \mathbb{R}^{k \times n \times l}$
3: for $\alpha_j \in \{\alpha_1, \ldots, \alpha_k\}$ do
4:     declare $X_{\alpha_j} \in \mathbb{R}^{n \times l}$
5:     for $v_i \in V$ do
6:         $ld_i \leftarrow$ Compute_LD($v_i, \alpha_j, \epsilon$) (see Algorithm 1)
7:         $X_{\alpha_j}[i, :] = ld_i$
8:     end for
9:     $X[j, :, :] = X_{\alpha_j}$
10: end for
11: // Train classifier with SGD
12: $f = H_2(H_1(X))$
13: $f \leftarrow SGD(f, Y_{train}, \ell)$
14: return $f$
```

Table II: Real-world datasets used throughout the experiments. Datasets listed in the upper part are used for multiclass classification, and datasets in the lower part are used for multilabel classification.

Dataset	Nodes	Edges	Classes
CORA	2'708	5'278	7
CITESEER	3'264	4'536	6
PUBMED	19'717	44'324	3
BLOGCATALOG	10'312	333'983	39
IMDB GERMANY	32'732	1'175'364	27

holds that $m << n$. Finally, training with SGD is again in $O(n)$.

IV. EVALUATION

We evaluate our approach by performing node classification and compare the quality in terms of accuracy for multiclass prediction tasks, respectively micro F_1 and macro F_1 scores for multilabel prediction tasks. For both tasks, we compare our models against the following approaches:

- *GCN$_1$_only_L*: a GCN [14] model that applies convolution on the label matrix. We exclude self-links and use one convolution layer, followed by a dense output layer [2]
- *GCN*: a GCN model [14] with randomly initialized node embeddings.
- *GAT*: the model from [31] with randomly initialized node embeddings. Therefore it is similar to *GCN*, with the only difference being that attention is used to compute the weights for the neighborhood aggregation.
- *DeepWalk*: the DeepWalk model as proposed in [25]
- *node2vec*: the node2vec model as proposed in [10]
- *Planetoid-G*: the Planetoid variant that does not use node attributes [34] [3]

For the multiclass problems, we additionally compare against the two-step label propagation approach proposed in [23] (*2-step LP*). Finally, we demonstrate the ability of our method to adapt to local label neighborhoods by using synthetic networks that were generated with stochastic blockmodels. Each of those datasets is designed to show the performance of our method on networks that have different local label correlation patterns within the local node neighborhoods.

A. Multiclass Classification.

We use the three benchmark graph datasets CORA, CITESEER and PUBMED [21], [26]. All datasets are split as suggested in [34], i.e., for labeled data our training sets contain 20 randomly selected instances per class, the test sets consist of 1'000 instances and the validation sets contain 500 instances, for each method. The remaining instances are used as unlabeled data. We compute the micro F_1 scores in 10 random data splits. Since the number of iterations for sampling the graph contexts and the label contexts for *Planetoid* are suggested only for the *CiteSeer* data set, we adapted these values relative to the number of nodes for each graph. For *node2vec*, we perform grid searches over the hyperparameters p and q with $p, q \in \{0.25, 0.5, 1.0, 2.0, 4.0\}$ and window size 10 as proposed by the authors. For the *GCN* and *GAT* models we ranged the number of layers from 1 to 3, with all hidden layers being of the same dimensionality in $\{16, 32, 64, 128\}$. For all models except *Planetoid* unless otherwise noted, we use one hidden layer with 16 neurons, the learning rate and training procedure are used as proposed in [14]. Regarding our models, we use $\alpha = [0.01, 0.5, 0.9]$ as values for the teleportation parameter and $\epsilon = 1e^{-5}$ as approximation threshold to compute the APPR vectors for each node. To train our models, we perform grid search over the learning rate parameter $lr \in \{1e^{-1}, 1e^{-2}, 1e^{-3}, 1e^{-4}\}$ and the batch size $b \in \{256, 512, 1024\}$. For the concatenation models we use 16 or 32 hidden neurons per APPR matrix in the hidden layer and report the best results over all model instances, i.e., simply concatenating the input before feeding it to the network, or concatenating the representation in the hidden layer and differentiating between shared weight matrices and independently trained weight matrices.

Table III: Accuracy values for each method and the three considered benchmark graph data sets.

Method	CORA	CITESEER	PUBMED
GCN$_1$_only_L	0.498±0.088	0.256±0.059	0.218±0.027
GCN	0.664±0.024	0.426±0.034	0.670±0.025
GAT	0.634±0.019	0.406±0.019	0.622±0.023
DeepWalk	0.700±0.019	0.477±0.022	0.726±0.019
node2vec	0.710±0.020	0.494±0.028	0.738±0.021
Planetoid-G	0.635±0.026	0.440±0.022	0.629±0.043
2-step LP	0.723±0.020	0.483±0.020	0.724±0.022
LD_AVG	0.779±0.015	**0.615±0.012**	0.762±0.012
LD_CONCAT	**0.806±0.012**	0.612±0.018	**0.775±0.012**

Table III shows the accuracy scores we achieved for the multiclass prediction task for each considered model on the three benchmark datasets *Cora*, *CiteSeer* and *Pubmed*. Our models improve the best results produced by *node2vec* which demonstrates that label distributions learned from variously scaled neighborhoods are indeed a useful source of information. The evaluation for *GCN$_1$_only_L* shows, especially for *Pubmed*, rather poor results. This is due to this model considering only the label distribution of a very local neighborhood (in fact one-hop neighbors). For the *Cora* and the *CiteSeer* networks, the improvements of the micro F_1 scores of *LD_CONCAT*, when comparing to the result of *node2vec*, are significant. Moreover, for all datasets, our models perform similarly, which shows that even our simple models with shared weights are able to match the performance of more complex models.

[2]We use only a single convolution layer due to the reason stated in Section II

[3]Unless stated differently, we use the parameter settings as suggested by the corresponding authors for all competitors. Except for minor adaptations, e.g., to include label information in the one layer GCN model or to make the Planetoid models applicable for multilabel prediction tasks, we use the original implementations as published by the corresponding authors.

B. Multilabel Classification.

The multilabel node classifications are performed on the two multilabel networks BLOGCATALOG [29] and IMDB GERMANY [7]. As the fraction of positive instances is relatively small for most of the classes, we use weighted cross-entropy as loss function. Therefore, the loss caused by erroneously classified positive instances is weighted higher. We use weight 10 in all our experiments. We report micro F_1 and macro F_1 scores to measure the quality of the considered methods. We compare our models to the attribute-less models that we already used for the multiclass experiments[4]. We split the data into training, validation and test set such that 70% of all nodes were used for training, 10% for validation to determine the best hyperparameter setting and 20% of the data were used as test set to evaluate the models' performance. As we observed that the sensitivity to the number of training labels is similar for all methods, we only report the results when using 70% of the instances for training. We show experimental results where we varied the number of training instances in Section IV-D. Note that we could not use stratified sampling splits for the experiments since we optimize for all classes simultaneously instead of using one-vs-rest classifiers[5]. The hyperparameter setting is as described above. For this set of experiments we ran each model 10 times on five different data splits. Due to the long runtime of *Planetoid-G* we trained this model only three times on two data splits.

The results for the *BlogCatalog* and the *IMDb Germany* networks are shown in Table IV. For the *BlogCatalog* network, only using the label information from the direct neighborhood of a node is not useful to infer its labels, c.f. GCN_1*_only_L*. However, incorporating label distributions of somewhat larger neighborhoods as for our models (again, we use the APPR matrix calculated for small values of α to determine the label distribution in neighborhoods that span more than 1-hop neighbors) seems to improve the results for the prediction task significantly. Compared to all competitors our models achieve better performance for both micro and macro F_1 scores. In fact, the best among our models achieves a significant gain over the best competitor, i.e., *node2vec*. For the *IMDb Germany* network, the labels in the 1-hop neighborhood are already very expressive. In this setting, our models outperform all of the competitors significantly, except for the *GAT* model that shows similar performance based on the micro F_1 score.

C. Effect of Multi-Scale Neighborhoods.

Figure 2a shows the impact of the α parameter, resp. the volume of the considered neighborhood, on the performance of the different models. As an example, we took the *BlogCatalog* network and ran our base model, i.e., $H_1 = q(X_\alpha W)$, with H_1 again describing the first hidden layer, q being the *ReLU* activation function and X_α being the neighborhood label distribution for a single value of α. Note that *LD_INDP* is the concatenation model that learns the weight matrices for each α-neighborhood independently and concatenates the resulting representations after the hidden layer (cf., Eq. 4), while *LD_SHARED* denotes the instance of the concatenation model that shares weights across α-neighborhoods (cf., Eq. 5). To demonstrate the superiority of combining label distributions from differently sized neighborhoods, we compare the micro F_1 scores achieved with single neighborhoods to the one achieved by using the combination of only three differently sized neighborhoods, i.e., for $\alpha = [0.01, 0.5, 0.9]$, in Figure 2b.

The results show two things: (1) that the native label distribution embeddings are quite sensitive to the choice of α, and (2) that in addition to selecting a single optimal scale, combining multiple scales further improves performance. Finally, we can summarize that in cases where no a priori knowledge about "good" and "bad" locality levels is given for a certain dataset, even a small set of α values which range from "very local" (α close to 1) to "spacious" (α close to 0) is already sufficient to get useful node representations due to the models optimizing the combination of neighborhoods.

D. Different Label Patterns

To prove the intuition that our method actually adapts to different local label patterns, we use synthetic networks. Analogously to [23] we apply the stochastic block model (SBM) [12] to generate networks with 8000 nodes and mean degree 15. We define three types of nodes (classes) for each network. The relationships between different classes can be seen in Figure 3. The first network models homophily relations (top left), the second models heterophily (top right), the third models both types of relationships for different classes (bottom left), and for the fourth graph, each class type is subdivided into different groups to model locally mixing patterns of heterophily and homophily (bottom right). We compare our approaches to *2-step LP* [23], which can model heterophily and homophily.

As can be seen in Figure 3, our approaches clearly outperform *2-step LP* on all networks. Thus, even if all classes follow the same pattern it is beneficial to *learn* how labels relate to each other rather than modeling it explicitly. It is particularly interesting that for the most difficult case our simplest model *LD_AVG* performs worse than our concatenation model. This demonstrates that it is indeed useful to explicitly capture label distributions at multiple scales for each node individually. Therefore, we conclude that considering different scales is especially useful if labels do not follow a single pattern.

V. CONCLUSION

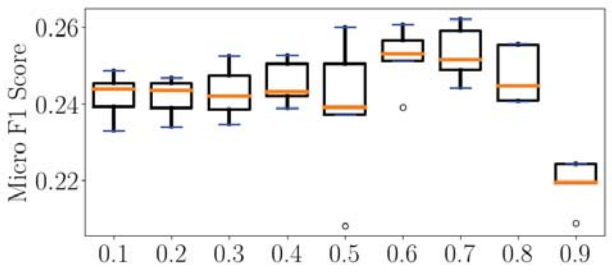
In this paper, we introduced Ada-LLD, a novel label-based approach to semi-supervised node classification in graphs. Our method aims at learning general correlations between a node's label and neighboring labels in an adaptive fashion. To learn from such correlations at multiple scales, we propose two different variants of our model both having a different inductive bias. Our experimental results on various real-work

[4]To adapt the *Planetoid-G* implementation for multilabel classification, we use a *sigmoid* activation function at the output layer.

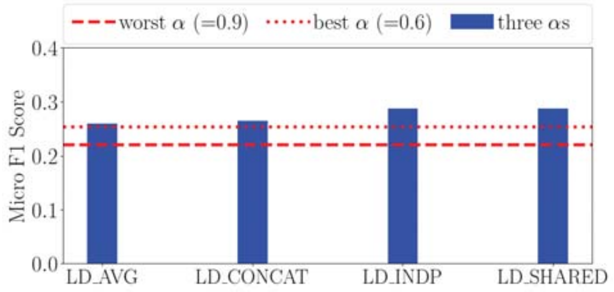
[5]That is why our results for *node2vec* and *DeepWalk* on the BlogCatalog network are slightly worse than reported in [10].

Table IV: Micro and Macro F_1 scores for each method and the two multilabel graph data sets.

Method	BlogCatalog		IMDb Germany	
	Micro F_1	Macro F_1	Micro F_1	Macro F_1
GCN_1_only_L	0.008±0.005	0.009±0.006	0.647±0.002	0.288±0.011
GCN	0.316±0.009	0.161±0.010	0.909±0.001	0.819±0.003
GAT	0.315±0.015	0.152±0.015	**0.915±0.007**	0.878±0.021
DeepWalk	0.286±0.007	0.113±0.006	0.404±0.012	0.303±0.004
node2vec	0.294±0.007	0.125±0.001	0.412±0.007	0.318±0.005
Planetoid-G	0.068±0.004	0.059±0.003	0.147±0.014	0.109±0.005
LD_AVG	0.324±0.005	**0.217±0.007**	0.860±0.010	0.862±0.011
LD_CONCAT	**0.349±0.007**	0.204±0.008	0.914±0.011	**0.918±0.009**



(a) Micro F1 scores for single α values.



(b) Comparing micro F1 scores of our models when using single α values (dashed lines) vs combining three different α values (bars).

Figure 2: Effect of differently sized neighborhoods for label distribution models on the BlogCatalog network.

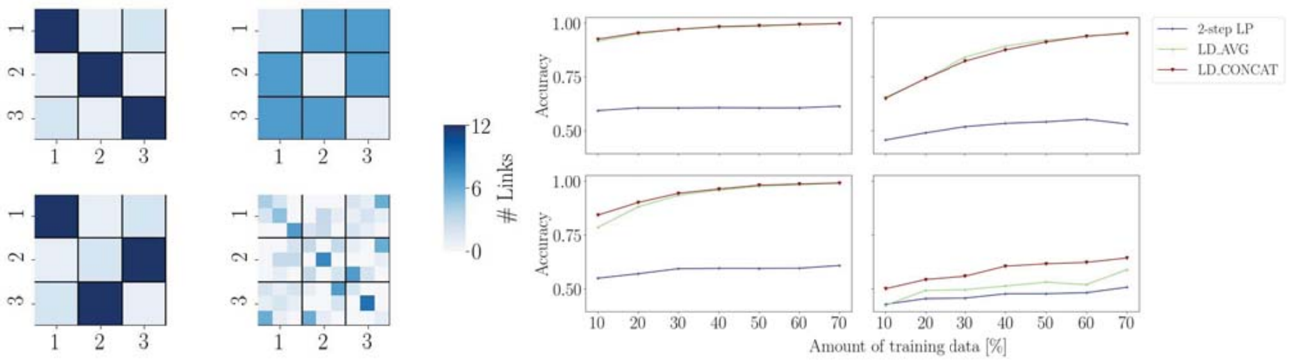


Figure 3: Block interaction matrices and corresponding results for different amounts of training instances. The color of the blocks denotes the average number of edges linking nodes from class i to class j.

datasets demonstrate that local label distributions are able to significantly improve node classification in multiclass as well as in multilabel settings. In addition, we show that Ada-LLD indeed profits from the label distributions in multiple neighborhoods. For future work, we plan to further investigate how to effectively combine label-based features with different other kinds of features, such as node attributes, edge attributes or node embeddings.

Acknowledgement

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibilities for its content.

References

[1] Atwood, J., Towsley, D.: Search-convolutional neural networks. CoRR **abs/1511.02136** (2015), `http://arxiv.org/abs/1511.02136`

[2] Berkhin, P.: Bookmark-coloring algorithm for personalized pagerank computing. Internet Mathematics **3**(1), 41–62 (2006)

[3] Bojchevski, A., Günnemann, S.: Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. arXiv preprint arXiv:1707.03815 (2017)

[4] Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs. CoRR **abs/1312.6203** (2013), `http://arxiv.org/abs/1312.6203`

[5] Cao, S., Lu, W., Xu, Q.: Grarep: Learning graph representations with global structural information. In: Proc. of CIKM. pp. 891–900. ACM (2015)

[6] Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29, pp. 3844–3852 (2016)

[7] Faerman, E., Borutta, F., Fountoulakis, K., Mahoney, M.W.: Lasagne: Locality and structure aware graph node embedding. arXiv preprint arXiv:1710.06520 (2017)

[8] Gatterbauer, W., Günnemann, S., Koutra, D., Faloutsos, C.: Linearized and single-pass belief propagation. VLDB Endowment **8**(5), 581–592 (2015)

[9] Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. arXiv preprint arXiv:1704.01212 (2017)

[10] Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proc. of ACM SIGKDD. pp. 855–864 (2016)

[11] Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NIPS. pp. 1025–1035 (2017)

[12] Holland, P.W., Laskey, K.B., Leinhardt, S.: Stochastic blockmodels: First steps. Social networks **5**(2), 109–137 (1983)

[13] Jeh, G., Widom, J.: Scaling personalized web search. In: Proc. of the 12th WWW. pp. 271–279. ACM (2003)

[14] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)

[15] Koutra, D., Ke, T.Y., Kang, U., Chau, D.H.P., Pao, H.K.K., Faloutsos, C.: Unifying guilt-by-association approaches: Theorems and fast algorithms. In: Proc. of ECML PKDD. pp. 245–260. Springer (2011)

[16] Levie, R., Monti, F., Bresson, X., Bronstein, M.M.: Cayleynets: Graph convolutional neural networks with complex rational spectral filters. CoRR **abs/1705.07664** (2017)

[17] Li, R., Wang, S., Zhu, F., Huang, J.: Adaptive graph convolutional neural networks. arXiv preprint arXiv:1801.03226 (2018)

[18] Mikael Henaff, Joan Bruna, Y.L.: Deep convolutional networks on graph-structured data. arXiv preprint arXiv:1506.05163 (2015)

[19] Misra, V., Bhatia, S.: Bernoulli embeddings for graphs. arXiv preprint arXiv:1803.09211 (2018)

[20] Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., Bronstein, M.M.: Geometric deep learning on graphs and manifolds using mixture model cnns. CoRR **abs/1611.08402** (2016), `http://arxiv.org/abs/1611.08402`

[21] Namata, G., London, B., Getoor, L., Huang, B., EDU, U.: Query-driven active surveying for collective classification. In: Workshop on Mining and Learning with Graphs (2012)

[22] Pearl, J.: Reverend Bayes on inference engines: A distributed hierarchical approach. Cognitive Systems Laboratory, School of Engineering and Applied Science, University of California, Los Angeles (1982)

[23] Peel, L.: Graph-based semi-supervised learning for relational networks. In: Proc. of SDM. pp. 435–443. SIAM (2017)

[24] Peel, L., Delvenne, J.C., Lambiotte, R.: Multiscale mixing patterns in networks. Proceedings of the National Academy of Sciences **115**(16), 4057–4062 (2018)

[25] Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proc. of ACM SIGKDD. pp. 701–710 (2014)

[26] Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. AI magazine **29**(3), 93 (2008)

[27] Shun, J., Roosta-Khorasani, F., Fountoulakis, K., Mahoney, M.W.: Parallel local graph clustering. Proc. VLDB Endow. **9**(12), 1041–1052 (Aug 2016)

[28] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: Proc. of WWW. pp. 1067–1077. ACM (2015)

[29] Tang, L., Liu, H.: Relational learning via latent social dimensions. In: Proc. of ACM SIGKDD. pp. 817–826. ACM (2009)

[30] Tsitsulin, A., Mottin, D., Karras, P., Müller, E.: Verse: Versatile graph embeddings from similarity measures. In: Proc. of WWW. pp. 539–548 (2018)

[31] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)

[32] Wang, B., Tu, Z., Tsotsos, J.K.: Dynamic label propagation for semi-supervised multi-class multi-label classification. In: Proceedings of the IEEE international conference on computer vision. pp. 425–432 (2013)

[33] Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: Proc. of ACM SIGKDD. pp. 1225–1234. ACM (2016)

[34] Yang, Z., Cohen, W., Salakhudinov, R.: Revisiting semi-supervised learning with graph embeddings. In: Proc. of ICDM. pp. 40–48 (2016)

[35] Zhang, Z., Cui, P., Wang, X., Pei, J., Yao, X., Zhu, W.: Arbitrary-order proximity preserved network embedding. In: Proc. of SIGKDD. pp. 2778–2786 (2018)

[36] Zhou, D., Bousquet, O., Lal, T.N., Weston, J., Schölkopf, B.: Learning with local and global consistency. In: Proc. of NIPS. pp. 321–328 (2004)

[37] Zhu, X., Ghahramani, Z.: Learning from labeled and unlabeled data with label propagation (2002)

[38] Zhu, X., Ghahramani, Z., Lafferty, J.D.: Semi-supervised learning using gaussian fields and harmonic functions. In: Proc. of ICML. pp. 912–919 (2003)

Towards Learning Structural Node Embeddings using Personalized PageRank

Felix Borutta, Julian Busch, Evgeniy Faerman, Matthias Schubert

Ludwig-Maximilians-Universität München

In this abstract we present our work in progress on embeddings for nodes in graphs representing their structural similarities. Intuitively, the more similar the structural representations of the respective neighbors of two nodes, the more structurally similar they are. An exemplary application is given by the task of identifying the role of each node in the graph, which might for instance correspond to the function of a protein within a protein-protein interaction network. Modeling and developing formal definitions for concepts such as structural similarity, structural identity and roles remain challenging problems. Existing works either learn embeddings which are not able to appropriately model structural similarity or identify roles based on hand-crafted features rather than learning important features directly from the input graph. Recent work[1] proposes a framework for learning a structural embedding vector for each node based on the degree sequences of it's neighbors within a k-hop distance. Two major drawbacks of this approach are the cubic time complexity and the fact that different k-hop neighborhoods are considered equally important. Note that in most cases the closer neighbors are considered more important. In contrast, we compute Personalized PageRank (PPR) for each node in the graph and use the resulting vectors as node representations. PPRs are very fast to compute and effectively capture the probability distribution over the corresponding neighborhoods. The preliminary results in Table 1 prove our intuition: we simply sorted the raw PPR vector of each node and used them as representations. However, the experiments used in[1] are of small practical relevance, as they use strongly artificial datasets. Therefore, we also work on an appropriate evaluation framework.

Network	complete graph				removed 30% of edges			
	struc2vec		ppr		struc2vec		ppr	
	corresp.	all	corresp.	all	corresp.	all	corresp.	all
Karate	$.012 \pm .007$	$.517 \pm .275$	$.0 \pm .0$	$.082 \pm .048$	$.381 \pm .208$	$.532 \pm .245$	$.102 \pm .098$	$.209 \pm .149$
Barbell	$.008 \pm .005$	$.171 \pm .178$	$.0 \pm .0$	$.066 \pm .069$	$.012 \pm .008$	$.166 \pm .212$	$.041 \pm .055$	$.159 \pm .136$
PPI	–	–	$.0 \pm .0$	$.402 \pm .225$	–	–	$.081 \pm .098$	$.405 \pm .229$

Table 1. Experimental results on three mirrored networks with each having one bridging edge; the graph setting and setup for struc2vec is the same as in [1]. 'corresp.' shows the mean cosine distances $\pm$ the standard deviation between the node embeddings of the corresponding nodes, 'all' shows the corresponding values when the distance is measured pairwise between all embeddings. Note that struc2vec did not terminate after 3 weeks for the mirrored PPI network (7780 nodes, 77479 edges).

[1] struc2vec: Learning Node Representations from Structural Identity. Figueiredo, Daniel R and Ribeiro, Leonardo FR and Saverese, Pedro HP. accepted for KDD'17

Structural Graph Representations based on Multiscale Local Network Topologies

Felix Borutta
Ludwig-Maximilians-Universität München
borutta@dbs.ifi.lmu.de

Julian Busch[*]
Ludwig-Maximilians-Universität München
busch@dbs.ifi.lmu.de

Evgeniy Faerman[*]
Ludwig-Maximilians-Universität München
faerman@dbs.ifi.lmu.de

Adina Klink
Ludwig-Maximilians-Universität München
klink@cip.ifi.lmu.de

Matthias Schubert
Ludwig-Maximilians-Universität München
schubert@dbs.ifi.lmu.de

ABSTRACT

In many applications, it is required to analyze a graph merely based on its topology. In these cases, nodes can only be distinguished based on their structural neighborhoods and it is common that nodes having the same functionality or role yield similar neighborhood structures. In this work, we investigate two problems: (1) how to create structural node embeddings which describe a node's role and (2) how important the nodes' roles are for characterizing entire graphs. To describe the role of a node, we explore the structure within the local neighborhood (or multiple local neighborhoods of various extents) of the node in the vertex domain, compute the visiting probability distribution of nodes in the local neighborhoods and summarize each distribution to a single number by computing its entropy. Furthermore, we argue that the roles of nodes are important to characterize the entire graph. Therefore, we propose to aggregate the role representations to describe whole graphs for graph classification tasks. Our experiments show that our new role descriptors outperform state-of-the-art structural node representations that are usually more expensive to compute. Additionally, we achieve promising results compared to advanced state-of-the-art approaches for graph classification on various benchmark datasets, often outperforming these approaches.

KEYWORDS

Representation Learning, Node Embedding, Graph Classification, Structural Embeddings

ACM Reference Format:
Felix Borutta, Julian Busch[*], Evgeniy Faerman[*], Adina Klink, and Matthias Schubert. 2019. Structural Graph Representations based on Multiscale Local Network Topologies. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI '19), October 14–17, 2019, Thessaloniki, Greece.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3350546.3352505

1 INTRODUCTION

The increasing relevance of graph-structured data has been accompanied by an increased interest in algorithms that can leverage underlying graph structure to make accurate predictions about the modeled entities. In many scenarios no additional facts about entities or properties of relationships are known. In such cases, the only source of information for machine learning tasks like node and graph classification is the graph topology. In this work, we consider two problems, i.e., deriving structural node representations (or *role representations*) based solely on the topological structure within the local node neighborhoods, and using these role representations to learn representations for entire graph structures in an unsupervised manner.

First, we consider the problem of deriving role representations for nodes based on the topological structure within the local neighborhood. The idea is to capture the different functionalities of network entities within vector representations such that entities that play a similar role within the network end up close together in the embedded space. Note that the notion of a *role* is generally diverse and might describe influencers in a social network, or a specific group of atoms in molecule networks that are likely to bind to similar atomic substructures. In general, node representations describing the roles of the nodes within a graph are useful for downstream classification or clustering tasks, e.g., they may give valuable insights for real world tasks like drug design, identification of influencers within social communities, link prediction, etc. To introduce the problem more formally, we are given a set of graphs $\mathcal{G} = \{G_1, \ldots, G_N\}$ with $G_i = (V_i, E_i)$ being a graph, $V = \bigcup_{i=1}^{N} V_i$ denoting the set of vertices and $E = \bigcup_{i=1}^{N} E_i$ being the set of edges. The goal is to derive vector representations $f(v_j) \in \mathbb{R}^d$ reflecting the various roles of nodes $v_j \in V$ in a graph $G_i \in \mathcal{G}$. Figure 1 illustrates the concept of node roles for airline networks. Note that the color coding corresponds to the nodes' roles. It can clearly be seen that the role of a node (e.g. hub airport, airport with connection to a single hub or an airport with connection to several hubs) can be extracted from the node's local neighborhood. We can also see that these roles can be identified across different graphs although the local neighborhoods may seem to be different, e.g., in terms of size. However, considering the local topology around the nodes, it can also be seen that they are similar for those nodes

[*] These authors contributed equally.

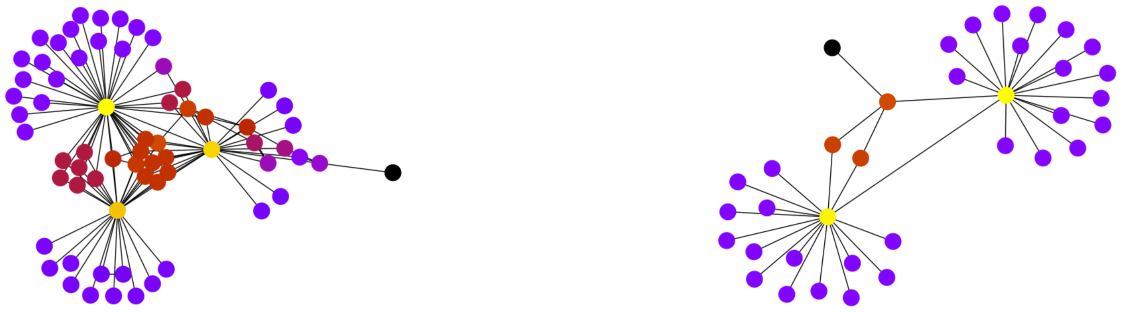


Figure 1: Airline networks. Left: Scandinavian Airline; Right: Niki Air. Each node corresponds to an airport and edges connect two airports if the airline operates a flight between them. The color coding corresponds to the role descriptors as determined by our approach.

that have similar colors. The distinct property of node role representations is that they should be independent of specific neighbors. Therefore, nodes that are similar in the embedded space are not necessarily closely connected and even may reside in different graphs. In particular, given two nodes $u, v \in V$ which have similar local structural neighborhood patterns with respect to some similarity measure, i.e., $S_N(u) \approx S_N(v)$, then the representation of u and v shall be similar as well. However, defining an appropriate similarity measure seems difficult since even the notion of local structural neighborhood patterns is hard to grasp. In this paper, we argue that the spread of probability mass under the node's most relevant local neighbors is a good characteristic for the node's role. Similarly to [11] we leverage the *Approximate Personalized PageRank (APPR)* to effectively describe multiple locality structures around the vertices and use the probability distribution vectors as a basis to quantify the structural roles of the nodes. An important feature of our novel node representation is that it is very efficient to compute and thus, even suitable for large data sets.

The second task considered in this work is to learn vector representations for entire graphs based on the nodes' roles. Calculating numerical vector representations for entire graph structures is particularly useful when considering tasks like graph classification. Many applications, especially in biology or sociology, may benefit from well-suited graph descriptors, e.g., for deciding whether a particular molecule is toxic or not, or for identifying similar social groups. Formally, for graph classification, we are given a label vector $y \in \{1, \ldots, K\}^N$ assigning each graph $G_i \in \mathcal{G}$ to one of K classes. The goal is to learn a model, which accurately predicts the class label y_i given G_i. By using an unsupervised clustering algorithm, we discretize the definition of roles and represent each graph by a count vector for role types.

Our empirical evaluation demonstrates that our simple approach outperforms advanced state-of-the-art role-based node representations. Furthermore, we show that even with straightforward aggregation, we already outperform state-of-the-art approaches for graph classification on the majority of the used evaluation datasets. To summarize, the main contributions of this paper are as follows:

- A novel structure-based approach to determine role representations for single nodes directly in the vertex domain.
- A novel approach for graph classification based on aggregation of node roles appearing in a graph.
- An extensive evaluation of our proposed role representations as well as the derived graph representations.

2 RELATED WORK

A variety of approaches have been proposed for solving different problems related to our work. These can mainly be sorted into two categories, *node embedding* and *graph classification* techniques.

2.1 Node Embedding

Node embedding techniques aim at deriving continuous vector representations for nodes in a graph based on certain assumptions. These techniques are unsupervised and the learned embeddings can be incorporated for several downstream learning tasks.

Homophily-based approaches embed nodes such that nodes which are close in the graph should also be close in the resulting embedding space and mainly differ in how they determine node neighborhoods. While some methods, e.g., the ones presented in [11, 13, 26, 32] rely on implicit matrix factorization, other methods such as *GraRep* [8] perform explicit matrix factorization. More recent methods focus on additional aspects like incorporating node attributes [14] or uncertainty [5]. However, though they also learn node representations in general, their objective is to embed nodes such that the vector representations capture proximity rather than structural properties.

In contrast, the idea of structural node embeddings is to relate nodes if their connectivity patterns to their respective neighbors exhibit similar structural properties. To this end, *struc2vec* [27] trains a SkipGram model using degree sequences in neighborhoods of increasing size. However, the method hardly scales in terms of graph size. Similarly, *DRNE* [34] sorts neighboring nodes by their degree and feeds the resulting sequences into an LSTM. The authors show that in some special cases, the resulting embeddings satisfy regular equivalence which recursively defines two nodes in a graph to be role-equivalent if their neighbors have the same roles. While such graph-based role definitions are rather strict and often do not apply in the real world, we take a more flexible feature-based embedding approach. *RolX* [15] is a feature-based approach which relies on handcrafted structural features (such as node degree or clustering coefficient) and computes soft-assignments of nodes to a predefined number of roles using matrix factorization. For a more in-depth discussion on role discovery, we refer to [28].

A diffusion-based approach is taken by *GraphWave* [10], where the graph is first transformed to the spectral domain in which the signal is then filtered with a heat kernel. Approximation using Chebyshev polynomials of order up to k results in linear time complexity and a k-localized filter. Node embeddings are derived by aggregating the rows of the resulting wavelet coefficient matrix while controlling the spread of the diffusion implied by the heat kernel. For detecting patterns at multiple scales, embeddings resulting from different scaling parameters are computed and concatenated. Though the heat kernel diffusion process resembles that implied by PPR [9], an important difference is that our method operates directly in the vertex domain. Furthermore, our method is not restricted to k-hop neighborhoods.

2.2 Graph Classification

Methods for graph classification can be categorized into *kernel-based* and *feature-based* methods.

Established graph kernels include the *Random-Walk (RW)* [7], *Shortest-Path (SP)* [6], *Weisfeiler-Lehman (WL)* [29] and *Graphlet Kernel (GK)* [30]. The first two methods rely on additional node labels and count visits to nodes with a certain label in random walks or via shortest paths, respectively. The WL kernel is based on the Weisfeiler-Lehman graph isomorphism test and performs a relabeling of initial node labels. The graphlet kernel counts occurrences of small induced non-isomorphic subgraphs and does not consider additional node labels. A problem of those kernels is that correlations between feature dimensions are not taken into account. This problem is addressed, e.g., in [1, 21, 35] by learning hidden representations of the substructures counted by the respective graph kernels. However, it should be noted that all of the above methods suffer from rather high complexity. Other works focus on directly specifying graph metric spaces. Such approaches are usually NP-hard and rely on heuristics [2, 24].

Earlier works on feature-based graph classification rely on handcrafted features. *NetSimile* [4] extracts structural features (such as node degree and clustering coefficient) for each node and aggregates them per graph using different aggregation functions. Similar to graphlet kernels, other works focus on describing graphs by decomposing them into subgraphs. *Subgraph2vec* [21] computes subgraph embeddings using a SkipGram model. The more recent method *GE-FSG* [22] represents graphs as bags of frequent subgraphs and learns graph embeddings using a document embedding technique. *GAM* [18] addresses the problems of scalability and noise by using an attention model to focus on small and informative parts of a graph. However, the method relies on additional node attributes.

DeepGraph [19] computes graph embeddings based on heat kernel signatures. However, the final embeddings are trained end-to-end for predicting network growth. In addition to the heat kernel signature, *NetLSD* [33] further considers the wave kernel signature. Similarly as in GraphWave, signatures of different scales are concatenated in order to obtain a multi-scale representation and a k-th order approximation is performed to make the eigendecomposition of the Laplacian scalable. Message Passing Networks [12] is a class of neural networks models for graphs. The primary focus of these methods lies on learning embeddings from node attributes and they cannot be applied out-of-the-box to classify general non-attributed

Algorithm 1 APPR-Roles

```
Input: Graph $\mathcal{G}$, Labels $\mathscr{L}$, Teleportation probabilities $\alpha$s, Approximation
    threshold $\epsilon$
Output: Classification model $m$
 1: role_descriptors = list()
 2: for idx in range($\mathcal{G}$) do
 3:     v = $\mathcal{G}$.getNode(idx)
 4:     $emb_v$ = list()
 5:     for $\alpha$ in $\alpha$s do
 6:         $p_v^\alpha$ = APPR($v$, $\alpha$, $\epsilon$)
 7:         $emb_v$.append(entropy($p_v^\alpha$))
 8:     end for
 9:     role_descriptors.append($emb_v$)
10: end for
11: $m$ = LogisticRegression().fit(role_descriptors, $\mathscr{L}$)
12: return $m$
```

graphs. *Patchy-san* [23] addresses this problem by considering auxiliary node labels such as degree or PageRank centrality.

3 APPR ROLES: STRUCTURAL NODE AND GRAPH REPRESENTATIONS

Since our approach relies on the Approximate Personalized PageRank for capturing structural properties within local node neighborhoods, we briefly review its idea.

In general, Personalized PageRank can be viewed as a special case of the *PageRank* algorithm [25], where the probabilities in the starting vector are biased towards some set of nodes. We consider the special case in which the starting vector is a unit vector, resulting in personalized importance scores for the particular source node. Doing this for all nodes in the graph, the PPR-vectors finally can be stored as rows of a sparse PPR-matrix $\Pi \in \mathbb{R}^{n \times n}$. Local push-based algorithms [3, 16] can be used to compute *Approximate Personalized PageRank (APPR)* very efficiently and lead to sparse solutions where small, irrelevant entries are omitted. In particular, we consider the algorithm proposed in [31]. The algorithm requires two parameters to be set by the user. The teleportation parameter α determines the effective size of the neighborhood considered for the source node. The second parameter ϵ is a threshold which controls approximation quality and runtime. Note that the complexity for this procedure is $O(1/\alpha\epsilon)$.

3.1 Entropy-based Node Descriptors

The APPR-vector p_i of a node v_i effectively models the connectivity of that node with respect to all other nodes in the graph as a probability distribution, whereby the probability mass is concentrated only on v_i's relevant neighbors. In principle, we could use the APPR-vectors directly as node representations. This would lead to the following feature space:

$$\Delta^n = \left\{ p \in \mathbb{R}^n_{\geq 0} \;\middle|\; \sum_{i=1}^{n} p_i = 1 \right\}, \tag{1}$$

which is known as the n-dimensional standard simplex. However, the resulting representations model homophily rather than structural properties, since they encode the information to which individual nodes a particular source node is connected. In order to make the representations location-invariant, we need to factor out this information. Since location invariance in this case translates

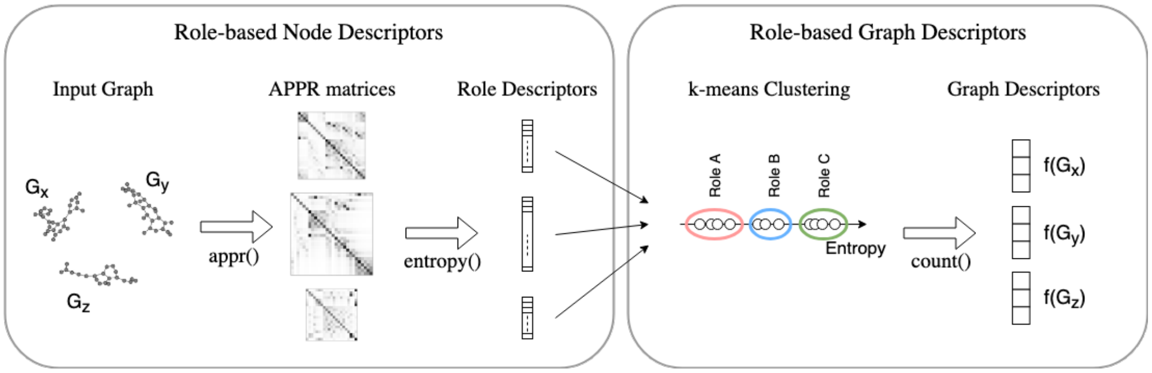


Figure 2: Workflow for calculating the aggregated graph descriptors from role-based node descriptors.

to permutation invariance, we consider the quotient space

$$\Delta^n \big/ \sim \; = \left\{[p] \;\middle|\; p \in \Delta^n\right\}, \tag{2}$$

which corresponds to the set of equivalence classes $[p] = \{q \in \Delta^n \mid p \sim q\}$ of the equivalence relation $\sim$ with $p \sim q \Leftrightarrow \exists P \in \mathbb{P} : p = qP$ where $\mathbb{P} = \left\{P \in \{0,1\}^{n \times n} \;\middle|\; P1 = 1,\; P^T 1 = 1\right\}$ is the set of permutation matrices. As a corresponding quotient map, we can define $f : \Delta^n \to \Delta^n \big/ \sim$ with $f(p) = pP_p$ which maps p to its equivalence class by sorting it using the permutation matrix P_p such that $1 \geq f(p)_1 \geq \cdots \geq f(p)_n \geq 0$.

Though the resulting sorted APPR-vectors qualify as structural node descriptors, they are not well suited for further downstream tasks since they are high-dimensional and sparse. Furthermore, node descriptors would not be comparable among graphs with different numbers of nodes. To this end we need to perform some form of aggregation. Our approach is based on the observation that, in terms of APPR, the structural properties of two nodes differ mostly based on the extent to which they spread their probability mass throughout the graph. For instance, a community node will spread its probability mass evenly to nodes within the same community, whereas a peripheral node will strongly concentrate its probability mass to one or very few nodes to which it is connected. The above behavior can be accurately described by the *Shannon entropy* $H : \Delta^n \big/ \sim \; \to \mathbb{R}$ with $H(p) = -\sum_{i=1}^{n} p_i \log p_i$ where we use the binary logarithm. In particular, it fulfills the following properties.

Theorem 3.1. *For all $p \in \Delta^n \big/ \sim$ it holds that*

(1) $H(p) \in [0, \log n]$.
(2) $H(p) = 0$ *if and only if* $p = e_1$.
(3) $H(p) = \log n$ *if and only if* $p = \frac{1}{n}1$.
(4) $H(p) = \log n - D_{KL}(p \parallel \frac{1}{n}1)$.

Intuitively stated, properties (1) to (3) state that the entropy is minimized for a distribution with a single peak and maximized for the uniform distribution. Property (4) states that the entropy can be interpreted as the similarity to the uniform distribution in terms of the *Kullback-Leibler divergence*. Our empirical results support the usefulness of this intuition. A further advantage over other applicable dimension reduction techniques is that we can describe each node by a single scalar value which can be visualized directly on a color map (as was done in Figure 1) and has a simple and intuitive meaning. Note that the entropy function is symmetric, i.e., $H(f(p)) = H(p)$ for all $p \in \Delta^n$. As a result, the APPR-vectors need not be sorted and the entropy of a single node v_i can be computed in linear time with respect to the number of non-zero entries of p_i.

Recalling that the teleportation parameter α in APPR controls the effective neighborhood size, we detect roles on multiple scales by computing APPR for multiple parameter values $\alpha \in \{\alpha_1, \ldots, \alpha_l\}$ and concatenating for each node its corresponding l entropy values. The final descriptor of node v_i is then given as

$$f_i = \left[H\left(p_i^{(\alpha_1)}\right), \ldots, H\left(p_i^{(\alpha_l)}\right)\right]^T \in \mathbb{R}^l. \tag{3}$$

The entire procedure for calculating the role descriptors can be summarized as outlined in Algorithm 1. The input for the method is the entire graph dataset $\mathcal{G}$, a list of node labels $\mathcal{L}$, a list of teleportation parameters αs, and the approximation threshold ϵ. For each of the nodes v in $\mathcal{G}$, the algorithm stacks the entropy-based representations of the corresponding APPR vectors, denoted as p_v^{α}, to generate the role descriptor of v, i.e., emb_v. The algorithm finally fits a classification model on the collection of role descriptors and retrieves the resulting model for node classification.

3.2 Aggregated Graph Descriptors

Since our structural node descriptors are location-invariant and thus transferable among different graphs, we are able to compare whole graphs by comparing their respective sets of node descriptors. However, simply collecting the node descriptors in a (ordered) set for each graph is not a straightforward solution, since this would result in different length representations for graphs with different numbers of nodes. To this end, we propose the following aggregation scheme to discretize the notion of roles: First we collect node descriptors from all graphs in the training dataset and cluster them with k-Means [20]. The resulting cluster centers $\{\mu_i \in \mathbb{R}^l \mid i = 1, \ldots, k\}$ can be interpreted as multi-scale role concepts appearing in the dataset. In a second step, we assign each node v in a given graph G_i to its nearest cluster center $\mu(v)$ and use the resulting count vector

$$F_i = \left[\left|\left\{v \in V_i \mid \mu(v) = \mu_j\right\}\right| : j = 1, \ldots, k\right]^T \in \mathbb{R}^k, \tag{4}$$

as representation for that graph. One important advantage of these graph descriptors is that they can be computed very efficiently, i.e., in linear time w.r.t. the total number of nodes in the dataset. Furthermore, the number of clusters k can be varied flexibly to explore different numbers of roles in a graph. For a supervised

Dataset	$\|\mathcal{G}\|$	$\|\mathcal{L}\|$	$\phi\|V\|$	$\phi\|E\|$
MUTAG	188	2	17.93	19.79
ENZYMES	600	6	32.63	62.14
NCI1	4110	2	29.87	32.30
NCI109	4127	2	29.68	32.13
PROTEINS	1113	2	39.06	72.82
IMDB-BINARY	2000	2	429.63	497.75
IMDB-MULTI	1500	3	13.00	65.94
REDDIT-BINARY	2000	2	429.63	497.75
REDDIT-12K	11929	11	391.41	456.89
REDDIT-5K	4999	5	508.52	594.87

Table 1: Benchmark datasets for graph classification. The upper part of the table contains biological networks, the lower part of the table refers to social network datasets. $|\mathcal{G}|$ denotes the number of graphs, $|\mathcal{L}|$ is the number of classes and $\phi|V|$, resp. $\phi|E|$ is the average number of nodes, resp. edges.

objective, the hyper-parameter can simply be optimized over a range of sensible values. However, other clustering techniques may be employed for discretizing the continuous role descriptors, too.

Figure 2 visualizes the workflow for calculating the described graph descriptors. The procedure consists of two blocks: in the first block, the continuous role descriptors for each node are calculated. Given the raw network – composed of multiple, differently sized components which form graph structures on their own – as input, we compute the stationary APPR distributions for each node. Next, we derive the continuous role-based node descriptors by computing the entropy values of the distributions for each node. Stacking these entropy values for each component results in differently sized and thus incomparable vectors (or matrices in case of multiple α values for the calculations of the APPR distributions). In order to enable comparisons between differently sized subgraphs, we first discretize the notion of roles by employing the k-means algorithm on the continuous role descriptors in the second block of our procedure[1]. Secondly, for each of the subgraph structures, we count the appearances of each role within the corresponding network to construct equally-sized graph descriptors which can easily be used for downstream tasks like classifications. Note that the example depicts the procedure for a single value of α used for APPR. As we show in the experiments section, richer representations can be calculated by using multiple values for α.

4 EXPERIMENTS

In the following, we first investigate the performance of our new node representations compared to node representations created with RolX [15] which relies on hand-crafted features, the diffusion-based method GraphWave [10] and the SkipGram-based struc2vec [27] method. Afterwards, we evaluate the proposed graph representations in graph classification tasks. We compare our approach against state-of-the-art approaches including Deep Graph Kernels (DGK) [35], the diffusion-based NetLSD [33] method and NetSimile [4] which relies on hand-crafted features.

[1]In the figure, we used $k = 3$ as the number of roles.

4.1 Datasets

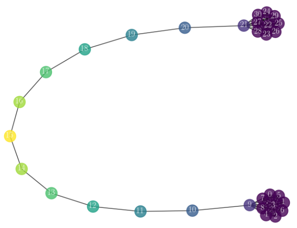
To empirically investigate the benefits of our approach for determining the role descriptors, we first use a barbell graph to compare the performance of various structure based node representations. The barbell graph consists of two fully connected components that are connected by a long chain. Specifically, we use a barbell graph that has ten nodes within each clique and a chain of length ten as illustrated in Figure 3a. The goal of this experiment is to demonstrate the ability of identifying different roles. In a second experiment, we use the mirrored Karate network as already used in [10, 27]. In particular, this dataset consists of two copies of the Zachary's karate network with one edge connecting the two copies by linking one randomly chosen node with its copy. We use this dataset to demonstrate the performance of the embedding techniques in terms of accuracy. Obviously, a node and its copy should end up at the same position (or very close to each other) in the embedding space as they have exactly the same roles. Finally, we employ two real world datasets, i.e., the European and the American air traffic networks used in [27]. These networks are unweighted and undirected. Nodes represent airports which are connected if there have been commercial flights during the time this dataset was recorded. Both networks have four equally sized classes corresponding to the relative level of activity of the airport. These datasets are used to compare the performance in terms of accuracy as well as to demonstrate the efficiency in terms of computation time.

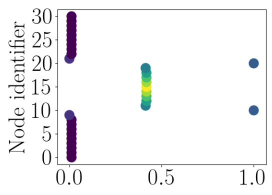
For evaluating the performance of the proposed graph representations in terms of graph classification, we use several biological and social networks which are taken from [17] (see Table 1).

4.2 Structural Node Embedding

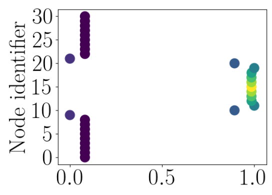
We first consider the structural node embeddings and compare the embeddings retrieved by our approach to the representations retrieved by RolX, GraphWave and struc2vec. For all competitors, we use the implementations and recommended configurations as published by the authors. The α parameter for our approach is ranged from 0.1 to 0.9 with a step size of 0.1. For each of the parameters, we get a one dimensional descriptor for every node, referred to as APPRroles$_{\alpha=i}$. We also construct higher dimensional representations by combining the node descriptors for all values of α per node. We refer to this configuration as APPRroles$_{stacked}$. The approximation threshold ϵ for the computations of the APPR vectors is set to $\epsilon = 1E^{-4}$.

Barbell Graph. In Figure 3 we provide a visual analysis of how well the APPR based node representations are able to embed the structural properties of the node neighborhoods. Figure 3a depicts the barbell graph while the remaining plots show 1-dimensional structural node embeddings. For visualization purposes, we project higher dimensional embeddings into 1-dimensional spaces by using PCA to be able to discuss the outcomes in comparison to our 1-dimensional representations. Therefore, we use the node identifiers on the y-axes in our plots to spread the depicted 1-dimensional embeddings along the y-axes such that they do not cover each other. Figures 3b-3d show the structural node embeddings for the competitors when projecting them onto the first principal component and normalizing the values (cf. x-axes). The first three images in the lower row depict the results for our approach when using different values for α. Precisely, we show the normalized results for

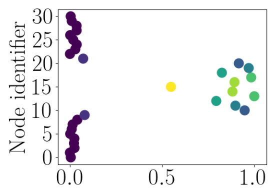

(a) Barbell Graph



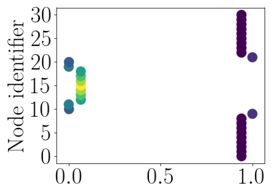
(b) RolX



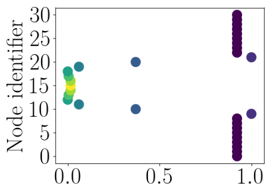
(c) GraphWave



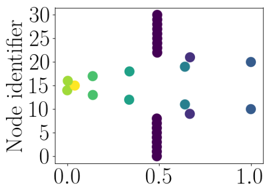
(d) struc2vec



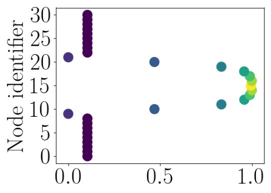
(e) APPRroles$_{\alpha=0.9}$



(f) APPRroles$_{\alpha=0.5}$



(g) APPRroles$_{\alpha=0.1}$



(h) APPRroles$_{stacked}$

Figure 3: 1-dimensional node representations for the barbell graph. The y-axes show the node identifiers (i.e., numerical values that we assigned to the nodes), the x-axes are the 1-dimensional node descriptors on [0,1] scale.

$\alpha \in \{0.9, 0.5, 0.1\}$, which means that we range the exploration of the node neighborhoods from local ($\alpha = 0.9$) to spacious ($\alpha = 0.1$). In Figure 3h, we visualize the node embeddings when stacking the descriptors for all values of α and projecting these embeddings into 1-dimensional space, again using PCA.

The first thing we want to emphasize is that it is difficult to say which of the methods works best on this dataset since we do not use any external evaluation measures in this experiment due to lack of ground truth. What we can clearly see is that RolX and GraphWave reveal only very few roles, i.e., most nodes of the chain are considered to have the same role. This also applies to struc2vec as we can identify three clusters in the 1-dimensional projection. However, differently to the other methods, struc2vec identifies the node in the center of the chain as an own role. Given our representations, we can see that our approach is much more flexible in terms of role identification. When using a rather large value for α, i.e., defining roles only based on very local neighborhoods, our method also considers the chain elements to have the same role. However, when decreasing the value of α, i.e., enlarging the neighborhoods based on which to define the roles, we can observe that the roles of the elements in the chain are considered to differ more and more. Nevertheless, the chain nodes that have the same hop distance from the center of the chain are always considered to have an equal role. Given that roles are not always defined precisely, this is a very desirable property of our approach that RolX or struc2vec cannot fulfill. GraphWave might be able to have this flexibility but due to operating on the spectra of the graphs rather than the vertex domain, it might be difficult to set the corresponding parameter appropriately.

Summarizing the insights revealed by this experiment, we can state that the choice of the value for α might have a significant impact on the outcome. In other words, using a small α value leads to an accurate distinction between node roles, while rather large values of α do not distinguish as accurately between different roles.

Mirrored Karate Network. Next, we consider the mirrored Karate network for which we measure the performance of the embedding methods by doing 1-NN-range queries. Recall that this network consists of two copies of the Zachary's Karate network and an additional edge which links a randomly chosen node with its copy. Given a set of structural node embeddings $\mathcal{E}$ containing one embedding for each node of the mirrored karate network, this experiment aims at identifying the copy of the query node among its 1-nearest-neighbors. Note that we call this query 1-NN-range query to emphasize that the set of 1-nearest-neighbors might be of size greater than 1. Precisely, we compute the nearest neighbor o for each query point $q \in \mathcal{E}$ and subsequently perform a range query around q with distance $dist(o, q)$.

The results given in Table 2 show the accuracy and average size of the queries' candidate sets, i.e., $\varnothing|\mathcal{C}|$. Except for struc2vec, all methods achieve an accuracy of 100%. However, when considering the precision that we measure by the size of the candidate set of the 1-NN-range query, we can see that our approach gives the best result on average.

Method	Acc.	$\varnothing\|\mathcal{C}\|$	$\frac{Acc.}{\varnothing\|\mathcal{C}\|}$
RolX	1.00	6.59	0.151
GraphWave	1.00	6.56	0.152
struc2vec	0.56	3.00	0.186
APPRroles$_{\alpha=0.1}$	0.96	4.53	0.211
APPRroles$_{\alpha=0.2}$	**1.00**	**4.62**	**0.216**
APPRroles$_{\alpha=0.3}$	0.99	5.62	0.175
APPRroles$_{\alpha=0.4}$	1.00	5.00	0.200
APPRroles$_{\alpha=0.5}$	1.00	5.44	0.183
APPRroles$_{\alpha=0.6}$	1.00	4.91	0.203
APPRroles$_{\alpha=0.7}$	0.93	4.35	0.212
APPRroles$_{\alpha=0.8}$	1.00	5.76	0.173
APPRroles$_{\alpha=0.9}$	1.00	6.32	0.158
APPRroles$_{stacked}$	1.00	4.97	0.201

Table 2: Results for the 1-NN-range queries on the mirrored Karate network. $\varnothing|\mathcal{C}|$ denotes the average size of the candidate sets retrieved by the 1-NN-range queries.

Method	EUR Airports	t in ms	USA Airports	t in ms
RolX	0.78±0.06	16747	0.77±0.05	66463
GraphWave	0.78±0.05	16596	0.77±0.04	263405
struc2vec	0.81±0.09	695616	0.84±0.05	830.94
APPRroles$_{\alpha=0.1}$	0.77±0.03	375	0.75±0.00	772
APPRroles$_{\alpha=0.2}$	0.76±0.03	337	0.75±0.01	530
APPRroles$_{\alpha=0.3}$	0.75±0.01	209	0.76±0.03	431
APPRroles$_{\alpha=0.4}$	0.75±0.01	.9	0.78±0.04	379
APPRroles$_{\alpha=0.5}$	0.77±0.04	189	0.78±0.05	331
APPRroles$_{\alpha=0.6}$	0.77±0.04	153	0.78±0.05	307
APPRroles$_{\alpha=0.7}$	0.76±0.03	143	0.79±0.07	281
APPRroles$_{\alpha=0.8}$	0.74±0.03	134	0.78±0.06	240
APPRroles$_{\alpha=0.9}$	0.75±0.00	50	0.78±0.06	252
APPRroles$_{stacked}$	**0.79±0.07**	**1788**	**0.80±0.07**	**3523**

Table 3: Results for one-vs-rest classification. We report the mean accuracy and standard deviation for each configuration. The table also reports runtime measurements for creating the representations including preprocessing steps.

Airport Traffic Networks. The results for the two airport traffic networks can be reviewed in Table 3. As proposed in [27] we employ one-vs-rest classifications with 90-10 train-test splits and report the mean accuracy and standard deviation over 10 runs. Additionally, the table contains the runtimes in milliseconds for each method including the preprocessing steps for struc2vec and our approach. RolX and GraphWave do not have preprocessing steps if executing their standard configurations[2].

For both networks, we can notice that although our proposed method outperforms RolX and GraphWave in terms of accuracy, the scores achieved with struc2vec are slightly better. Note that the results for the European airports network are comparable across all methods. However, regarding the runtime our method clearly outperforms the competitors. Compared to struc2vec our approach is more than 2'300 times faster, even when considering the APPRroles$_{stacked}$ configuration, on the USA airports network[3]. It should be noted that the accuracy of our approach might be further improved by spending more time for preprocessing the local neighborhoods. As we use an approximate version of the Personalized PageRank for engineering the local neighborhoods, it is possible to decrease the approximation threshold to get more accurate node descriptors. However, this comes at the cost of increased computation time.

4.3 Graph Classification

We next show that the proposed node representations are well-suited for graph classification when aggregating them for entire graph structures. Therefore, we compare our graph representations against Deep Graph Kernels (DGK), NetLSD and NetSimile. Though the authors of DGK present three kernel types, i.e., graphlet, shortest path and Weisfeiler-Lehmann kernels, the latter two require additional node labels which are considered by none of the other methods. For the sake of a fair comparison, we only compare against graphlet kernels. For NetLSD, we use both presented configurations, i.e., the variant that uses heat trace signatures, denoted as NetLSD$_{Heat}$, as well as the variant that uses wave trace signatures, denoted as NetLSD$_{Wave}$. Either way, we calculate the full eigenspectra rather than using one of the proposed approximations. Regarding our approach, we present the results for $\alpha = 0.1$ and the variant where we stack the representations for all values of α.

For all experiments in this section, we report the mean accuracy of 1-NN classifications over 10 runs. The results for DGK are borrowed from the original paper since the method relies on a Kernel-SVM with precomputed kernels and the authors used the same evaluation setup.

Biological Networks. The results for the biological datasets are listed in Table 4. On the first four datasets our approach achieves better accuracy scores than the competitors. Indeed our approach reaches gains of approx. 3% on MUTAG, 2.8% on ENZYMES, about 8.2% on NCI1 and over 10% on NCI109 compared to the best competitor. For PROTEINS, the deep graph kernel method shows the best accuracy. However, among the methods that we evaluate with our classifier, our method achieves the best score.

Social Networks. Finally, considering the social networks, we can observe similar results. As can be seen in Table 5, except for IMDB-BINARY, our method outperforms all competitors for which we applied the 1-NN classification. Though we want to note that the deep graphlet kernel was slightly better on IMDB-MULTI when using the evaluation method proposed in [35].

5 CONCLUSION

In this work, we presented a novel approach for defining structural node representations that describe the nodes' roles within a graph. We subsequently used these node representations to compute meaningful vector representations for entire graph structures. More precisely, we figured out that local, personalized PageRank distributions encapsulate the structure of local node neighborhoods and can be compressed to meaningful role descriptors. Compared to previous approaches that tackle the problem of structural node embeddings our approach is fast to compute and allows for much flexibility in terms of identifying different node roles by simultaneously being quite simple to interpret. We show that these role descriptors are expressive enough to achieve good performance in terms of graph classification when aggregating them for entire

[2]Note that the standard configuration of GraphWave implements an automatic mode for parameter selection. Thus, we must compare its runtime to APPRroles$_{stacked}$.
[3]The EUR Airports network consists of 399 nodes and 5995 edges. The USA Airports network consists of 1190 nodes and 13599 edges.

Method	MUTAG	ENZYMES	NCI1	NCI109	PROTEINS
DGK	0.827±0.021	0.271±0.008	0.625±0.003	0.627±0.002	0.717±0.005
NetLSD$_{Heat}$	0.841±0.032	0.367±0.042	0.665±0.014	0.647±0.018	0.649±0.023
NetLSD$_{Wave}$	0.809±0.055	0.265±0.042	0.611±0.012	0.603±0.016	0.603±0.028
NetSimile	0.833±0.034	0.387±0.046	0.676±0.019	0.663±0.018	0.602±0.023
PPRroles$_{\alpha=0.1}$	**0.871±0.043**	0.340±0.033	0.704±0.019	0.699±0.014	**0.665±0.042**
PPRroles$_{stacked}$	0.858±0.043	**0..9±0.046**	**0.732±0.012**	**0.734±0.013**	0.644±0.028

Table 4: Accuracy in 1-NN classification on biological networks. Note that the DGK results are taken from [35] and report the achieved classification accuracy when applying a SVM.

Method	IMDB-BINARY	IMDB-MULTI	REDDIT-BINARY	REDDIT-12K	REDDIT-5K
DGK	0.670±0.006	0.446±0.005	0.780±0.004	0.322±0.001	0.413±0.002
NetLSD$_{Heat}$	0.690±0.034	0.421±0.035	0.782±0.017	0.246±0.009	0.332±0.014
NetLSD$_{Wave}$	0.681±0.033	0.420±0.038	0.671±0.023	0.227±0.005	0.317±0.012
NetSimile	**0.704±0.039**	0.415±0.039	0.848±0.010	0.335±0.009	0.413±0.012
PPRroles$_{\alpha=0.1}$	0.645±0.032	0.404±0.041	0.857±0.017	0.348±0.009	**0.427±0.014**
PPRroles$_{stacked}$	0.666±0.035	**0.429±0.039**	**0.867±0.014**	**0.348±0.007**	0.417±0.010

Table 5: Accuracy in 1-NN classification on social networks. Note that the DGK results are taken from [35] and report the achieved classification accuracy when applying a SVM.

graph structures. Our experiments demonstrate that we can outperform state-of-the-art methods in both node classification as well as graph classification tasks.

ACKNOWLEDGEMENTS

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibilities for its content.

REFERENCES

[1] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. 2018. Sub2Vec: Feature Learning for Subgraphs. In *Proc. of PAKDD*. Springer, 170–182.

[2] Jose Bento and Stratis Ioannidis. 2018. A Family of Tractable Graph Distances. In *Proc. of SDM*. 333–341.

[3] Pavel Berkhin. 2006. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics* 3, 1 (2006), 41–62.

[4] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. 2012. Netsimile: A scalable approach to size-independent network similarity. *arXiv preprint 1209.2684* (2012).

[5] Aleksandar Bojchevski and Stephan Günnemann. 2017. Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815* (2017).

[6] Karsten M Borgwardt and Hans-Peter Kriegel. 2005. Shortest-path kernels on graphs. In *Proc. of ICDM*. 8–pp.

[7] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl_1 (2005), i47–i56.

[8] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proc. of CIKM*. ACM, 891–900.

[9] Fan Chung. 2007. The heat kernel as the pagerank of a graph. *Proceeddings of the National Academy of Sciences* 104, 50 (2007), 19735–19740.

[10] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. 2018. Learning structural node embeddings via diffusion wavelets. In *Proc. of KDD*. ACM, 1320–1329.

[11] Evgeniy Faerman, Felix Borutta, Kimon Fountoulakis, and Michael W Mahoney. 2018. Lasagne: Locality and structure aware graph node embedding. In *Proc. of WI)*. IEEE, 246–253.

[12] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *Proc. of ICML*. JMLR. org, 1263–1272.

[13] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proc. of KDD*. 855–864.

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proc. of NIPS*. 1024–1034.

[15] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. 2012. Rolx: structural role extraction & mining in large graphs. In *Proc. of KDD*. ACM.

[16] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *Proc. of WWW*. ACM, 271–279.

[17] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. 2016. Benchmark Data Sets for Graph Kernels. http://graphkernels.cs.tu-dortmund.de

[18] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. 2018. Graph Classification using Structural Attention. In *Proc. of KDD*. ACM, 1666–1674.

[19] Cheng Li, Xiaoxiao Guo, and Qiaozhu Mei. 2016. DeepGraph: Graph structure predicts network growth. *arXiv preprint arXiv:1610.06251* (2016).

[20] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[21] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. 2016. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv preprint arXiv:1606.08928* (2016).

[22] Dang Nguyen, Wei Luo, Tu Dinh Nguyen, Svetha Venkatesh, and Dinh Phung. 2018. Learning graph representation via frequent subgraphs. In *Proc. of SDM*.

[23] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *Proc. of ICML*. 2014–2023.

[24] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. 2017. Matching Node Embeddings for Graph Similarity.. In *Proc. of AAAI*. 2429–2435.

[25] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: bringing order to the web. (1999).

[26] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proc. of KDD*. 701–710.

[27] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *Proc. of KDD*. 385–394.

[28] Ryan A Rossi and Nesreen K Ahmed. 2015. Role discovery in networks. *IEEE TKDE* 27, 4 (2015), 1112–1131.

[29] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12, Sep (2011), 2539–2561.

[30] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*. 488–495.

[31] Julian Shun, Farbod Roosta-Khorasani, Kimon Fountoulakis, and Michael W. Mahoney. 2016. Parallel Local Graph Clustering. *Proc. of VLDB Endowment* 9, 12 (Aug. 2016), 1041–1052.

[32] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proc. of WWW*. ACM, 1067–1077.

[33] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alex Bronstein, and Emmanuel Müller. 2018. NetLSD: Hearing the Shape of a Graph. In *Proc. of KDD*.

[34] Ke Tu, Peng Cui, Xiao Wang, Philip S Yu, and Wenwu Zhu. 2018. Deep Recursive Network Embedding with Regular Equivalence. In *Proc. of KDD*. ACM, 2357–2366.

[35] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proc. of SIGKDD*. ACM, 1365–1374.

Learning Self-Expression Metrics for Scalable and Inductive Subspace Clustering

Julian Busch* **Evgeniy Faerman** **Matthias Schubert** **Thomas Seidl**

Ludwig-Maximilians-Universität München
Munich Center for Machine Learning (MCML)
`{busch, faerman, schubert, seidl}@dbs.ifi.lmu.de`

Subspace clustering has established itself as a state-of-the-art approach to clustering high-dimensional data. In particular, methods relying on the self-expressiveness property have recently proved especially successful. However, they suffer from two major shortcomings: First, a quadratic-size coefficient matrix is learned directly, preventing these methods from scaling beyond small datasets. Secondly, the trained models are transductive and thus cannot be used to cluster out-of-sample data unseen during training. Instead of learning self-expression coefficients directly, we propose a novel metric learning approach to learn instead a subspace affinity function using a siamese neural network architecture. Consequently, our model benefits from a constant number of parameters and a constant-size memory footprint, allowing it to scale to considerably larger datasets. In addition, we can formally show that out model is still able to exactly recover subspace clusters given an independence assumption. The siamese architecture in combination with a novel geometric classifier further makes our model inductive, allowing it to cluster out-of-sample data. Additionally, non-linear clusters can be detected by simply adding an auto-encoder module to the architecture. The whole model can then be trained end-to-end in a self-supervised manner. This work in progress reports promising preliminary results on the MNIST dataset. In the spirit of reproducible research, me make all code publicly available. [1] In future work we plan to investigate several extensions of our model and to expand experimental evaluation.

1 Introduction

Subspace clustering [Vidal, 2011] assumes the data to be sampled from a union of low-dimensional subspaces of the full data space. The goal is to recover these subspaces and to correctly assign each data point to its respective subspace cluster. As a state-of-the-art approach to clustering high-dimensional data, it enables a multitude of applications, including image segmentation [Ma et al., 2007, Yang et al., 2008], motion segmentation [Kanatani, 2001, Elhamifar and Vidal, 2009, Ji et al., 2016], image clustering [Ho et al., 2003, Elhamifar and Vidal, 2013] and clustering gene expression profiles [McWilliams and Montana, 2014]. For instance, face images of a subject under fixed pose and varying lighting conditions [Basri and Jacobs, 2003] or images of hand-written digits with different rotations, translations and other natural transformations [Hastie and Simard, 1998] have been shown to lie in low-dimensional subspaces.

Recently, *self-expressiveness*-based methods [Elhamifar and Vidal, 2009, Liu et al., 2010, Lu et al., 2012, Elhamifar and Vidal, 2013, Liu et al., 2013, Wang et al., 2013, Feng et al., 2014, Ji et al., 2014, Vidal and Favaro, 2014, Ji et al., 2015, You et al., 2016a] have proved especially successful. The main idea is that each point can be expressed by a linear combination of points from the same subspace. This property is used to learn a quadratic-size coefficient matrix from which cluster labels can be extracted in a post-processing step using spectral clustering. The quadratic number of parameters prevents these methods from scaling beyond small datasets and makes them transductive and thus

[1] `https://github.com/buschju/sscn`

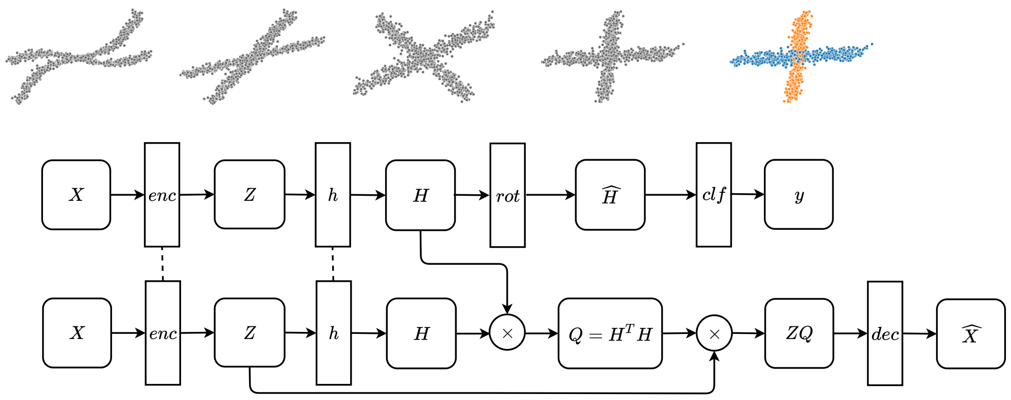


Figure 1: Data flow within our model. Square boxes denote tensors, rectangular boxes denote functions, and dashed lines indicate parameter sharing. After being mapped into a space in which the data fits better into independent linear subspaces using an encoder function enc, the data is mapped into a space using a function h in which independent clusters are orthogonal and dot-products corresponds to self-expression coefficients. Afterwards, the data is rotated into pre-defined axis-aligned subspaces and assigned to clusters based on the orthogonal projection distance (upper path). The original data is reconstructed from the self-expressed data in latent space by a decoder function dec to ensure that the learned embeddings are compatible with the original data (lower path).

inapplicable to out-of-sample data unseen during training. In contrast, our model requires only a constant number of parameters to provably provide the same expressive power. A classifier leveraging the unique geometric properties of our model further makes it inductive and enables it to cluster out-of-sample data.

In practice, we can usually not assume that the data exactly fits in linear subspaces, e.g., due to noise or additional non-linearities in the underlying data generating process which was not accounted for. Instead, we might rather assume the data to be situated on different *non-linear* sub-manifolds. Some methods [Chen and Lerman, 2009, Patel et al., 2013, Patel and Vidal, 2014, Xiao et al., 2016, Yin et al., 2016, Ji et al., 2017a] rely on the *kernel trick* to account for non-linearity. However, it is usually not clear whether a particular pre-defined kernel function is particularly suitable for subspace clustering. More recent methods [Peng et al., 2016, Ji et al., 2017b, Zhou et al., 2018, Zhang et al., 2019a, Seo et al., 2019, Kheirandishfard et al., 2020] learn a suitable feature transformation explicitly in an *end-to-end* differentiable model. In particular, *Deep Subspace Clustering Networks (DSC-Net)* [Ji et al., 2017b] introduced the idea of modeling the coefficient matrix as a dense neural network layer, called self-expressive layer, and training it jointly with an auto-encoder. As a result, the encoder represents a feature transformation which has been optimized w.r.t. linear cluster structure in latent space. *Self-Supervised Convolutional Subspace Clustering Networks (S^{2}ConvSCN)* [Zhang et al., 2019a] learn an additional classifier which can be applied to out-of-sample data but still rely on the full coefficient matrix and spectral clustering. Our model on the other hand offers both, applicability to out-of-sample data and scalability.

Several works have addressed the challenge of *scalability* but are either only able to detect linear clusters [You et al., 2016b, Rahmani and Atia, 2017], rely on a k-means-like procedure which requires good initialization and is sensitive to outliers [Zhang et al., 2018] or still fully parametrize coefficient matrices which need to be re-learned from scratch for each new data batch and come with no theoretical guarantees [Zhang et al., 2019b]. In contrast, our model is suitable to detect non-linear clusters, can be trained end-to-end with back-propagation and provides a theoretical foundation.

In summary, we propose, to the best of our knowledge, the first metric learning approach to subspace clustering, which enables a quadratic reduction of the number of parameters and memory footprint compared to existing methods while maintaining theoretical performance guarantees. Our model is applicable to out-of-sample data, suitable to detect non-linear clusters and can be trained end-to-end with back-propagation.

2 Siamese Subspace Clustering Networks

In subspace clustering, we are given a set set of points $\{x_i\}_{i=1}^N \subseteq \mathbb{R}^{d_X}$ sampled from a union of subspaces $\{S_i\}_{i=1}^K$ of unknown dimensions and arranged as columns of a data matrix $X \in \mathbb{R}^{d_X \times N}$. The goal is to recover these subspaces and to correctly assign each data point to its respective subspace cluster. While there exist many different variants of self-expressive subspace clustering, we focus here on a relaxed noise-aware version of *Efficient Dense Subspace Clustering (EDSC)* [Ji et al., 2014]:

Definition 1 (Efficient Dense Subspace Clustering [Ji et al., 2014])**.**

$$\min_{C \in \mathbb{R}^{N \times N}} \frac{1}{2} \|C\|_F^2 + \frac{\lambda}{2} \|X - XC\|_F^2 . \tag{1}$$

where the i-th column of the $N \times N$ coefficient matrix C contains the coefficients for expressing x_i. Regularization of C ensures that x_i is expressed using only points from the same subspace. Given the learned coefficient matrix, cluster assignments can be extracted in a post-processing step by applying spectral clustering to the subspace affinity matrix $A = |C| + |C|^T$. The unique solution to this problem can be expressed in closed-form as the solution C^* of the linear system $\left(I + \lambda X^T X\right) C = \lambda X^T X$ [Ji et al., 2014]. Let $r := \operatorname{rank}(X) = \dim\left(\bigoplus_{i=1}^K S_i\right)$ denote the rank of X. In a noise-free setting, if the subspaces are *independent*, i.e., if $r = \sum_{i=1}^K \dim(S_i)$, then C^* is guaranteed to be block-diagonal with $C^*_{ij} = 0$ if x_i and x_j originate from different subspaces [Vidal et al., 2008]. The corresponding solution is called *subspace-preserving*.

The central idea of our approach is to view subspace clustering from a metric learning perspective. To this end, we employ a siamese neural network [Bromley et al., 1994] consisting of two identical branches with shared weights and mirrored parameter updates which is optimized such that dot-products in latent space correspond to self-expression coefficients:

Definition 2 (Siamese Dense Subspace Clustering)**.**

$$\begin{aligned} \min_{\theta_h} \quad & \frac{1}{2} \|Q\|_F^2 + \frac{\lambda}{2} \|X - XQ\|_F^2 \\ \text{s.t.} \quad & Q = H^T H, \quad H = h(X; \theta_h) \end{aligned} \tag{2}$$

where $Q \in \mathbb{R}^{N \times N}$ contains the self-expression coefficients corresponding to dot-products of the embeddings $H \in \mathbb{R}^{d_H \times N}$ computed by the embedding function h. Note that weight sharing leads to symmetric coefficient matrices. Even though the reduction of parameters compared to (1) is quadratic, we can show that this model is able to recover the exact solution to the original subspace clustering problem, even when h consists of only a single linear layer with a sufficient number of neurons. Note that (2) is convex in this case.

Theorem 1. *Let $h(X) = WX$, $W \in \mathbb{R}^{d_H \times d_X}$, $d_H \geq r$, then (2) attains its global minimum at $W^* = R\sqrt{\lambda\left(I - \lambda\left(\Sigma_r^{-2} + \lambda I\right)^{-1}\right)}U_r^T$ where $X = U_r \Sigma_r V_r^T$ is the reduced SVD of X and $R \in \operatorname{St}(d_H, r)$ is an arbitrary orthonormal matrix. The unique optimal coefficient matrix Q^* of (2) corresponds to the unique solution of (1).*

Above, $\operatorname{St}(n, p) = \left\{X \in \mathbb{R}^{n \times p} \mid X^T X = I\right\}$ for $n \geq p$ denotes the *Stiefel manifold* which is composed of all $n \times p$ orthonormal matrices. Since (2) leads to a well-studied optimal solution, it can be analyzed directly within existing theory. In particular, it is guaranteed that under the independence assumption and in a noise-free setting, (2) yields a subspace-preserving solution. Also note that we don't need to know the exact rank of X, it is sufficient to have an upper bound. Since $r \leq \sum_{i=1}^K \dim(S_i)$, we can simply estimate the number of clusters K and the maximum cluster dimension q and set $d_H = Kq$ and $R \in \operatorname{St}(d_H, d_H)$.

Since we can choose R arbitrarily from $\operatorname{St}(d_H, d_H)$ and still obtain the same optimal coefficient matrix Q^*, we are able to optimize R on the Stiefel manifold w.r.t. to a cluster assignment objective where we take advantage of the observation that points from independent clusters will have orthogonal embeddings in H. To this end, we compute rotated embeddings $\widehat{H} = RH$ and then classify points by assigning them to their closest subspace w.r.t. orthogonal projection distance and applying the

	ACC	ARI	NMI	#Parameters	GPU-Memory (GB)
DSC-Net	63.54 ± 0.00	57.42 ± 0.00	$\mathbf{72.34 \pm 0.00}$	$100,014,991$	2.71
SSCN	$\mathbf{67.98 \pm 3.40}$	$\mathbf{58.53 \pm 3.34}$	69.48 ± 2.38	$\mathbf{66,291}$ $(\mathbf{-99.93\%})$	$\mathbf{0.19}$ $(\mathbf{-92.96\%})$
SSCN-OoS	67.39 ± 3.38	57.10 ± 3.27	67.16 ± 2.34	$66,291$	0.19

Table 1: Results on the MNIST dataset. Upper part: Transductive clustering of the 10,000 test images. Lower part: Inductive clustering of the 60,000 out-of-sample training images using our previously trained model. Note that our model did not see these images during training and that DSC-Net does not support clustering out-of-sample data and would require more than 4.9B parameters and 39GB of GPU-memory to cluster the whole dataset. All results are aggregated over 10 independent runs with different random initializations. For better comparability, all models use the same pre-trained auto-encoder. Results for DSC-Net exhibit no variation since the model uses constant initialization.

softmin function: $y_{ij} = {}^{\exp\left(-\|\hat{h}_i - S_j S_j^T \hat{h}_i\|_2^2\right)}/_{\sum_{k=1}^{K} \exp\left(-\|\hat{h}_i - S_k S_k^T \hat{h}_i\|_2^2\right)}$. The subspaces are fixed a-priori to be axis-aligned and don't need to be optimized. The matrix R is optimized such that classifications agree with self-expression affinities. For now, we compute the coefficient matrix of the training set using our trained model and then apply the same post-processing as in [Ji et al., 2017b] to obtain pseudo-labels which are used to train the classifier using *cross-entropy* loss and the *Cayley-Adam* algorithm [Li et al., 2020]. In future work, we plan to employ a triplet-loss [Hermans et al., 2017] which does not rely on spectral clustering and to additionally optimize the remaining model parameters with feedback from the classifier.

To account for non-linearity, we can simply add an auto-encoder to our model with the task of mapping the original data into a d_Z-dim. latent space in which the data better fits in linear subspaces and additionally the independence assumption can be better satisfied. This non-linear transformation is learned together with the rest of the model. We formalize our complete model in Definition 3.

Definition 3 (Siamese Subspace Clustering Network (SSCN))**.**

$$\begin{aligned} \min_{\theta_e, \theta_d, \theta_h, R} \quad & \frac{1}{2}\|Q\|_F^2 + \frac{\lambda_1}{2}\|Z - ZQ\|_F^2 + \frac{\lambda_2}{2}\left\|X - \widehat{X}\right\|_F^2 + \lambda_3 \mathcal{L}_{clf}(R; X) \\ \text{s.t.} \quad & Q = H^T H, \quad H = h\left(Z; \theta_h\right), \\ & Z = enc\left(X; \theta_e\right), \quad \widehat{X} = dec\left(ZQ; \theta_d\right) \end{aligned} \tag{3}$$

Above, $Z \in \mathbb{R}^{d_Z \times N}$ are non-linear embeddings of the input X computed by the encoder function enc. After self-expression in latent space, X is reconstructed as $\widehat{X} \in \mathbb{R}^{d_X \times N}$ using dec, a decoder function matching enc. The reconstruction loss ensures that the learned embeddings are actually compatible with the original data and prevents trivial solutions. Note that training with the full data batch X would lead to materialization of the full $N \times N$ coefficient matrix Q. This is not an issue for our model, however, since it can be trained with mini-batches and thus scale to large datasets. The only requirements are that batches need to be sampled uniformly at random and that the batch-size needs to be sufficiently large so that we sample enough instances from each class on average and thus obtain a representative sample. An illustration of the data flow is provided in Figure 1.

3 Experiments

As a first proof of concept, we compare our model with *DSC-Net* [Ji et al., 2017b] on the *MNIST* dataset [LeCun, 1998]. By default, we use a small convolutional auto-encoder and the same parameter settings for both models wherever possible. For SSCN, we model h as a single linear layer without bias as motivated above and train with a batch-size of 1000. The results are summarized in Figure 1. We can see that our model provides competitive performance while drastically reducing the required number of model parameters and GPU-memory. Even large amounts of out-of-sample data can be clustered reliably without any memory overhead. All hyper-parameter values and complete code for reproducing the reported results are provided in our public code repository. In future work we plan to train our model end-to-end using a triplet-loss and additional feedback from the classifier to the encoder and self-expression module. We further plan to evaluate on more datasets, with different architectural choices and against more baselines.

Acknowledgments and Disclosure of Funding

This work has been partially funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibilities for its content.

References

R Basri and DW Jacobs. Lambertian reflectance and linear subspaces. *TPAMI*, 2003.

Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. *NeurIPS*, 1994.

Guangliang Chen and Gilad Lerman. Spectral curvature clustering (scc). *IJCV*, 2009.

Ehsan Elhamifar and René Vidal. Sparse subspace clustering. *CVPR*, 2009.

Ehsan Elhamifar and Rene Vidal. Sparse subspace clustering: Algorithm, theory, and applications. *TPAMI*, 2013.

Jiashi Feng, Zhouchen Lin, Huan Xu, and Shuicheng Yan. Robust subspace segmentation with block-diagonal prior. *CVPR*, 2014.

Trevor Hastie and Patrice Y Simard. Metrics and models for handwritten character recognition. *Statistical Science*, 1998.

Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

J Ho, Ming-Husang Yang, Jongwoo Lim, Kuang-Chih Lee, and D Kriegman. Clustering appearances of objects under varying illumination conditions. *CVPR*, 2003.

Pan Ji, Mathieu Salzmann, and Hongdong Li. Efficient dense subspace clustering. *WACV*, 2014.

Pan Ji, Mathieu Salzmann, and Hongdong Li. Shape interaction matrix revisited and robustified: Efficient subspace clustering with corrupted and incomplete data. *ICCV*, 2015.

Pan Ji, Hongdong Li, Mathieu Salzmann, and Yiran Zhong. Robust multi-body feature tracker: a segmentation-free approach. *CVPR*, 2016.

Pan Ji, Ian Reid, Ravi Garg, Hongdong Li, and Mathieu Salzmann. Adaptive low-rank kernel subspace clustering. *arXiv preprint arXiv:1707.04974*, 2017a.

Pan Ji, Tong Zhang, Hongdong Li, Mathieu Salzmann, and Ian Reid. Deep subspace clustering networks. *NeurIPS*, 2017b.

Ken-ichi Kanatani. Motion segmentation by subspace separation and model selection. *ICCV*, 2001.

Mohsen Kheirandishfard, Fariba Zohrizadeh, and Farhad Kamangar. Multi-level representation learning for deep subspace clustering. *WACV*, 2020.

Yann LeCun. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

Jun Li, Li Fuxin, and Sinisa Todorovic. Efficient riemannian optimization on the stiefel manifold via the cayley transform. *ICLR*, 2020.

Guangcan Liu, Zhouchen Lin, and Yong Yu. Robust subspace segmentation by low-rank representation. *ICML*, 2010.

Guangcan Liu, Zhouchen Lin, Shuicheng Yan, Ju Sun, Yong Yu, and Yi Ma. Robust recovery of subspace structures by low-rank representation. *TPAMI*, 2013.

Can-Yi Lu, Hai Min, Zhong-Qiu Zhao, Lin Zhu, De-Shuang Huang, and Shuicheng Yan. Robust and efficient subspace segmentation via least squares regression. *ECCV*, 2012.

Yi Ma, Harm Derksen, Wei Hong, and John Wright. Segmentation of multivariate mixed data via lossy data coding and compression. *TPAMI*, 2007.

Brian McWilliams and Giovanni Montana. Subspace clustering of high-dimensional data: a predictive approach. *DMKD*, 2014.

Vishal M Patel and René Vidal. Kernel sparse subspace clustering. *ICIP*, 2014.

Vishal M Patel, Hien Van Nguyen, and René Vidal. Latent space sparse subspace clustering. *ICCV*, 2013.

Xi Peng, Shijie Xiao, Jiashi Feng, Wei-Yun Yau, and Zhang Yi. Deep subspace clustering with sparsity prior. *IJCAI*, 2016.

Mostafa Rahmani and George Atia. Innovation pursuit: A new approach to the subspace clustering problem. *ICML*, 2017.

Junghoon Seo, Jamyoung Koo, and Taegyun Jeon. Deep closed-form subspace clustering. *ICCV Workshops*, 2019.

René Vidal. Subspace clustering. *IEEE Signal Processing Magazine*, 2011.

René Vidal and Paolo Favaro. Low rank subspace clustering (lrsc). *Pattern Recognition Letters*, 2014.

René Vidal, Roberto Tron, and Richard Hartley. Multiframe motion segmentation with missing data using powerfactorization and gpca. *IJCV*, 2008.

Yu-Xiang Wang, Huan Xu, and Chenlei Leng. Provable subspace clustering: When lrr meets ssc. *NeurIPS*, 2013.

Shijie Xiao, Mingkui Tan, Dong Xu, and Zhao Yang Dong. Robust kernel low-rank representation. *IEEE transactions on neural networks and learning systems*, 2016.

Allen Y Yang, John Wright, Yi Ma, and S Shankar Sastry. Unsupervised segmentation of natural images via lossy data compression. *CVIU*, 2008.

Ming Yin, Yi Guo, Junbin Gao, Zhaoshui He, and Shengli Xie. Kernel sparse subspace clustering on symmetric positive definite manifolds. *CVPR*, 2016.

Chong You, Chun-Guang Li, Daniel P Robinson, and René Vidal. Oracle based active set algorithm for scalable elastic net subspace clustering. *CVPR*, 2016a.

Chong You, Daniel Robinson, and René Vidal. Scalable sparse subspace clustering by orthogonal matching pursuit. *CVPR*, 2016b.

Junjian Zhang, Chun-Guang Li, Chong You, Xianbiao Qi, Honggang Zhang, Jun Guo, and Zhouchen Lin. Self-supervised convolutional subspace clustering network. *CVPR*, 2019a.

Tong Zhang, Pan Ji, Mehrtash Harandi, Richard Hartley, and Ian Reid. Scalable deep k-subspace clustering. *ACCV*, 2018.

Tong Zhang, Pan Ji, Mehrtash Harandi, Wenbing Huang, and Hongdong Li. Neural collaborative subspace clustering. *ICML*, 2019b.

Pan Zhou, Yunqing Hou, and Jiashi Feng. Deep adversarial subspace clustering. *CVPR*, 2018.